
RHG Compute Tools Documentation

Release 999

Rhodium Group

Jun 15, 2023

Contents

1	RHG Compute Tools	3
1.1	Installation	3
1.2	Features	3
2	Installation	5
2.1	Stable release	5
2.2	From sources	5
3	Usage	7
3.1	Design tools	7
4	API	9
4.1	rhg_compute_tools package	9
5	Contributing	35
5.1	Types of Contributions	35
5.2	Get Started!	36
5.3	Pull Request Guidelines	37
5.4	Tips	37
6	History	39
6.1	Current version (unreleased)	39
6.2	v1.2.1	39
6.3	v1.2	39
6.4	v1.1.4	39
6.5	v1.1.3	39
6.6	v1.1.2	40
6.7	v1.1.1	40
6.8	v1.1	40
6.9	v1.0.1	40
6.10	v1.0.0	40
6.11	v0.2.2	40
6.12	v0.2.1	40
6.13	v0.2.0	41
6.14	v0.1.8	41
6.15	v0.1.7	41
6.16	v0.1.6	41

6.17	v0.1.5	41
6.18	v0.1.4	41
6.19	v0.1.3	42
6.20	v0.1.2	42
6.21	v0.1.1	42
6.22	v0.1.0	42
7	Indices and tables	43
	Python Module Index	45
	Index	47

Contents:

Tools for using `compute.rhg.com` and `compute.impactlab.org`

- Free software: MIT license
- Documentation: <https://rhg-compute-tools.readthedocs.io>.

1.1 Installation

pip:

```
pip install rhg_compute_tools
```

1.2 Features

1.2.1 Kubernetes tools

- easily spin up a preconfigured cluster with `get_cluster()`, or flavors with `get_micro_cluster()`, `get_standard_cluster()`, `get_big_cluster()`, or `get_giant_cluster()`.

```
>>> import rhg_compute_tools.kubernetes as rhgk
>>> cluster, client = rhgk.get_cluster()
```

1.2.2 Google cloud storage utilities

- Utilities for managing google cloud storage directories in parallel from the command line or via a python API

```
>>> import rhg_compute_tools.gcs as gcs
>>> gcs.sync_gcs('my_data_dir', 'gs://my-bucket/my_data_dir')
```

2.1 Stable release

To install RHG Compute Tools, run this command in your terminal:

```
$ pip install rhg_compute_tools
```

This is the preferred method to install RHG Compute Tools, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

2.2 From sources

The sources for RHG Compute Tools can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/RhodiumGroup/rhg_compute_tools
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/RhodiumGroup/rhg_compute_tools/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```


To use RHG Compute Tools in a project:

```
import rhg_compute_tools
```

3.1 Design tools

`rhg_compute_tools` includes various helper functions and templates so you can more easily create plots in Rhodium and Climate Impact Lab styles.

To use these tools, import the design submodule:

```
import rhg_compute_tools.design
```

When you do this, we'll automatically load the fonts, color palettes, and default plot specs into matplotlib. There are a couple ways you can use them:

3.1.1 Color schemes

Use one of the color schemes listed below as a `cmap` argument to a plot. For example, to use the `rhg_standard` color scheme in a bar chart:

```
>>> data = pd.DataFrame({
...     'A': [1, 2, 3, 2, 1],
...     'B': [2, 2, 1, 2, 2],
...     'C': [5, 4, 3, 2, 1]})
>>> data.plot(kind='bar', cmap='rhg_standard');
```

Rhodium Group Color Schemes

Categorical color schemes

- `rhg_standard`
- `rhg_light`

Continuous color schemes

- `rhg_Blues`
- `rhg_Greens`
- `rhg_Yellows`
- `rhg_Oranges`
- `rhg_Reds`
- `rhg_Purples`

You can also access the RHG color grid using the array `rhg_compute_tools.design.colors.RHG_COLOR_GRID`

4.1 rhg_compute_tools package

4.1.1 Subpackages

rhg_compute_tools.design package

Submodules

rhg_compute_tools.design.colors module

rhg_compute_tools.design.plotting module

rhg_compute_tools.design.plotting.**add_colorbar** (*ax*, *cmap*='viridis', *norm*=None, *orientation*='vertical', ***kwargs*)

Add a colorbar to a plot, using a pre-defined cmap and norm

Parameters

- **ax** (*object*) – matplotlib axis object
- **cmap** (*str* or *object*, *optional*) – matplotlib.colors.cmap instance or name of a registered cmap (default viridis)
- **norm** (*object*, *optional*) – matplotlib.colors.Normalize instance. default is a linear norm between the min and max of the first plotted object.
- **orientation** (*str*, *optional*) – default 'vertical'
- ****kwargs** – passed to colorbar constructor

rhg_compute_tools.design.plotting.**get_color_scheme** (*values*, *cmap*=None, *colors*=None, *levels*=None, *how*=None)

Generate a norm and color scheme from data

Parameters

- **values** (*array-like*) – data to be plotted, from which to generate cmap and norm. This should be an array, DataArray, etc. that we can use to find the min/max and/or quantiles of the data.
- **cmap** (*str, optional*) – named matplotlib cmap (default inferred from data)
- **colors** (*list-like, optional*) – list of colors to use in a discrete colormap, or with which to create a custom color map
- **levels** (*list-like, optional*) – boundaries of discrete colormap, provide
- **how** (*str, optional*) – Optional setting form {'linear', 'log', 'symlog', None}. Used to construct the returned norm object, which defines the way the colors map to values. By default, we the method is inferred from the values.

Returns

- **cmap** (*object*) – matplotlib.colors.cmap color mapping
- **norm** (*object*) – matplotlib.colors.Normalize instance using the provided values, levels, color specification, and “how” method

Module contents

rhg_compute_tools.design.get_color_scheme(values, cmap=None, colors=None, levels=None, how=None)

Generate a norm and color scheme from data

Parameters

- **values** (*array-like*) – data to be plotted, from which to generate cmap and norm. This should be an array, DataArray, etc. that we can use to find the min/max and/or quantiles of the data.
- **cmap** (*str, optional*) – named matplotlib cmap (default inferred from data)
- **colors** (*list-like, optional*) – list of colors to use in a discrete colormap, or with which to create a custom color map
- **levels** (*list-like, optional*) – boundaries of discrete colormap, provide
- **how** (*str, optional*) – Optional setting form {'linear', 'log', 'symlog', None}. Used to construct the returned norm object, which defines the way the colors map to values. By default, we the method is inferred from the values.

Returns

- **cmap** (*object*) – matplotlib.colors.cmap color mapping
- **norm** (*object*) – matplotlib.colors.Normalize instance using the provided values, levels, color specification, and “how” method

rhg_compute_tools.design.add_colorbar(ax, cmap='viridis', norm=None, orientation='vertical', **kwargs)

Add a colorbar to a plot, using a pre-defined cmap and norm

Parameters

- **ax** (*object*) – matplotlib axis object
- **cmap** (*str or object, optional*) – matplotlib.colors.cmap instance or name of a registered cmap (default viridis)

- **norm** (*object*, *optional*) – matplotlib.colors.Normalize instance. default is a linear norm between the min and max of the first plotted object.
- **orientation** (*str*, *optional*) – default ‘vertical’
- ****kwargs** – passed to colorbar constructor

rhg_compute_tools.io_tools package

Submodules

rhg_compute_tools.io_tools.readers module

```
rhg_compute_tools.io_tools.readers.get_fs(fp) → fsspec.spec.AbstractFileSystem
rhg_compute_tools.io_tools.readers.load_netcdf(blob, fsspec_kwargs=None, *args,
                                               retries=5, **kwargs) → xar-
                                               ray.core.dataset.Dataset
    Read a geotiff or raster file from a local or gs:// location
rhg_compute_tools.io_tools.readers.read_csv(blob, **fsspec_kwargs) → pan-
                                               das.core.frame.DataFrame
    Read a csv file from a local or gs:// location
rhg_compute_tools.io_tools.readers.read_csvv_response(fp) → xar-
                                               ray.core.dataset.Dataset
rhg_compute_tools.io_tools.readers.read_dataframe(blob, **fsspec_kwargs) → pan-
                                               das.core.frame.DataFrame
    Read a CSV or parquet file from a local or gs:// location
rhg_compute_tools.io_tools.readers.read_dataset(fp, engine=None, **kwargs) → xar-
                                               ray.core.dataset.Dataset
rhg_compute_tools.io_tools.readers.read_netcdf(blob, fsspec_kwargs=None, *args,
                                               retries=5, **kwargs) → xar-
                                               ray.core.dataset.Dataset
    Read a geotiff or raster file from a local or gs:// location. Very similar to load_netcdf, but without the load.
rhg_compute_tools.io_tools.readers.read_parquet(blob, **fsspec_kwargs) → pan-
                                               das.core.frame.DataFrame
    Read a parquet file from a local or gs:// location
rhg_compute_tools.io_tools.readers.read_rasterio(blob, fsspec_kwargs=None, *args,
                                                  retries=5, **kwargs) → xar-
                                                  ray.core.dataarray.DataArray
    Read a NETCDF file from a local or gs:// location
rhg_compute_tools.io_tools.readers.read_shapefile(fp)
rhg_compute_tools.io_tools.readers.read_zarr(fp, fs=None, mapper_kwargs=None,
                                              isel_dict: dict = None, load_data:
                                              bool = False, **kwargs) → xar-
                                              ray.core.dataset.Dataset
rhg_compute_tools.io_tools.readers.report_if_not_found(fp)
```

rhg_compute_tools.io_tools.writers module

```
rhg_compute_tools.io_tools.writers.document_dataset(ds: xarray.core.dataset.Dataset,
                                                    repo_root: str = '.') → xarray.core.dataset.Dataset

rhg_compute_tools.io_tools.writers.get_maximal_chunks_encoding(ds: xarray.core.dataset.Dataset,
                                                              **var_chunks)
                                                              → dict

rhg_compute_tools.io_tools.writers.write_netcdf(out_ds: xarray.core.dataset.Dataset,
                                                out_fp: str, fs: Union[None,
                                                                    fsspec.registry.filesystem] = None) →
                                                None

rhg_compute_tools.io_tools.writers.write_zarr(out_ds: xarray.core.dataset.Dataset,
                                              out_fp: str, fs: Union[None,
                                                                    fsspec.registry.filesystem] = None,
                                              set_maximal_chunks: bool = True,
                                              writer_kwargs: Optional[dict] = None,
                                              encoding_kwargs: Optional[dict] =
                                              None) → None
```

Module contents

4.1.2 Submodules

4.1.3 rhg_compute_tools.gcs module

Tools for interacting with GCS infrastructure.

```
rhg_compute_tools.gcs.authenticated_client(credentials=None, **client_kwargs)
```

Convenience function to create an authenticated GCS client.

Parameters

- **credentials** (*str* or *None*, *optional*) – Str path to storage credentials authentication file. If *None* is passed (default) will create a *Client* object with no args, using the authorization credentials for the current environment. See the [google cloud storage docs](<https://googleapis.dev/python/google-api-core/latest/auth.html>) for an overview of the authorization options.
- **client_kwargs** (*optional*) – kwargs to pass to the *get_client* function

Returns

Return type google.cloud.storage.Client

```
rhg_compute_tools.gcs.cp(src, dest, flags=[])
```

Copy a file or recursively copy a directory from local path to GCS or vice versa. Must have already authenticated to use. Notebook servers are automatically authenticated, but workers need to pass the path to the authentication json file to the `GOOGLE_APPLICATION_CREDENTIALS` env var. This is done automatically for `rhg-data.json` when using the `get_worker` wrapper.

Parameters

- **dest** (*src*,) – The paths to the source and destination file or directory. If on GCS, either the `/gcs` or `gs:/` prefix will work.

- **flags** (*list of str, optional*) – String of flags to add to the gsutil cp command. e.g. `flags=['r']` will run the command `gsutil -m cp -r...` (recursive copy)

Returns

- `str` – stdout from gsutil call
- `str` – stderr from gsutil call
- `datetime.timedelta` – Time it took to copy file(s).

`rhg_compute_tools.gcs.create_directories_under_blob(blob, project=None, client=None)`

`rhg_compute_tools.gcs.create_directory_markers(bucket_name, project=None, client=None, prefix=None)`

Add directory markers in-place on a google cloud storage bucket

Parameters

- **bucket_name** (*str*) – name of the Google Cloud Storage bucket
- **project** (*str or None, optional*) – name of the Google Cloud Platform project. If None, inferred from the default project as determined by the client.
- **client** (*google.cloud.storage.client.Client or None, optional*) – Optionally pass a google.cloud.storage Client object to set auth and settings
- **prefix** (*str or None, optional*) – Prefix (relative to bucket root) below which to create markers

`rhg_compute_tools.gcs.get_bucket(credentials=None, bucket_name='rhg-data', return_client=False, **client_kwargs)`

Return a bucket object from Rhg's GCS system.

Parameters

- **credentials** (*str or None, optional*) – Str path to storage credentials authentication file. If None is passed (default) will create a Client object with no args, using the authorization credentials for the current environment. See the [google cloud storage docs](<https://googleapis.dev/python/google-api-core/latest/auth.html>) for an overview of the authorization options.
- **bucket_name** (*str, optional*) – Name of bucket. Typically, we work with `rhg_data` (default)
- **return_client** (*bool, optional*) – Return the Client object as a second object.
- **client_kwargs** (*optional*) – kwargs to pass to the `get_client` function

Returns bucket

Return type `google.cloud.storage.bucket.Bucket`

`rhg_compute_tools.gcs.ls(dir_path)`

List a directory quickly using `gsutil`

`rhg_compute_tools.gcs.replicate_directory_structure_on_gcs(src, dst, client)`

Replicate a local directory structure on google cloud storage

Parameters

- **src** (*str*) – Path to the root directory on the source machine. The directory structure within this directory will be reproduced within `dst`, e.g. `/Users/myusername/my/data`

- **dst** (*str*) – A url for the root directory of the destination, starting with *gs://[bucket_name]/*, e.g. *gs://my_bucket/path/to/my/data*
- **client** (*google.cloud.storage.client.Client*) – An authenticated *google.cloud.storage.client.Client* object.

`rhg_compute_tools.gcs.rm(path, flags=[])`

Remove a file or recursively remove a directory from local path to GCS or vice versa. Must have already authenticated to use. Notebook servers are automatically authenticated, but workers need to pass the path to the authentication json file to the `GOOGLE_APPLICATION_CREDENTIALS` env var. This is done automatically for `rhg-data.json` when using the `get_worker` wrapper.

Parameters

- **path** (*str* or *pathlib.Path*) – The path to the source and destination file or directory. Either the */gcs* or *gs:/* prefix will work.
- **flags** (*list of str, optional*) – String of flags to add to the *gsutil rm* command. e.g. *flags=['r']* will run the command *gsutil -m rm -r...* (recursive remove)

Returns

- *str* – stdout from *gsutil* call
- *str* – stderr from *gsutil* call
- `datetime.timedelta` – Time it took to copy file(s).

`rhg_compute_tools.gcs.sync(src, dest, flags=['r', 'd'])`

Sync a directory from local to GCS or vice versa. Uses *gsutil rsync*. Must have already authenticated to use. Notebook servers are automatically authenticated, but workers need to pass the path to the authentication json file to the `GCPLOUD_DEFAULT_TOKEN_FILE` env var. This is done automatically for `rhg-data.json` when using the `get_worker` wrapper.

Parameters

- **dest** (*src,*) – The paths to the source and destination file or directory. If on GCS, either the */gcs* or *gs:/* prefix will work.
- **flags** (*list of str, optional*) – String of flags to add to the *gsutil cp* command. e.g. *flags=['r','d']* will run the command *gsutil -m cp -r -d...* (recursive copy, delete any files on dest that are not on src). This is the default set of flags.

Returns

- *str* – stdout from *gsutil* call
- *str* – stderr from *gsutil* call
- `datetime.timedelta` – Time it took to copy file(s).

4.1.4 rhg_compute_tools.kubernetes module

Tools for interacting with kubernetes.

`rhg_compute_tools.kubernetes.get_big_cluster(*args, **kwargs)`

Start a cluster with 2x the memory and CPU per worker relative to default

All arguments are optional. If not provided, defaults will be used. To view defaults, instantiate a `dask_gateway.Gateway` object and call `gateway.cluster_options()`.

Parameters

- **name** (*str, optional*) – Name of worker image to use (e.g. `rhodium/worker:latest`). If `None` (default), default to worker specified in `template_path`.
- **tag** (*str, optional*) – Tag of the worker image to use. Cannot be used in combination with `name`, which should include a tag. If provided, overrides the tag of the image specified in `template_path`. If `None` (default), the full image specified in `name` or `template_path` is used.
- **extra_pip_packages** (*str, optional*) – Extra pip packages to install on worker. Packages are installed using `pip install extra_pip_packages`.
- **profile** (*One of ["micro", "standard", "big", "giant"]*) – Determines size of worker. CPUs assigned are slightly under 1, 2, 4, and 8, respectively. Memory assigned is slightly over 6, 12, 24, and 48 GB, respectively.
- **cpus** (*float, optional*) – Set the CPUs requested for your workers as defined by `profile`. Will raise error if >7.5 , because our 8-CPU nodes need ~ 5 vCPU for kubernetes pods. (NOTE 12/15/20: This is currently set to 1 by default to allow for mapping big workflows across inputs, see <https://github.com/dask/dask-gateway/issues/364>).
- **cred_name** (*str, optional*) – Name of Google Cloud credentials file to use, equivalent to providing `cred_path='/opt/gcsfuse_tokens/{}.json'.format(cred_name)`. May not use if `cred_path` is specified.
- **cred_path** (*str, optional*) – Path to Google Cloud credentials file to use. May not use if `cred_name` is specified.
- **env_items** (*dict, optional*) – A dictionary of env variable ‘name’-‘value’ pairs to append to the env variables included in `template_path`, e.g.

```
{
    'MY_ENV_VAR': 'some string',
}
```

- **extra_worker_labels** (*dict, optional*) – Dictionary of kubernetes labels to apply to pods. `None` (default) results in no additional labels besides those in the template, as well as `jupyter_user`, which is inferred from the `JUPYTERHUB_USER`, or, if not set, the server’s hostname.
- **extra_pod_tolerations** (*list of dict, optional*) – List of pod toleration dictionaries. For example, to match a node pool NoSchedule toleration, you might provide:

```
extra_pod_tolerations=[
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    },
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org/dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    }
]
```

- **keep_default_tolerations** (*bool, optional*) – Whether to append (default) or replace the default tolerations. Ignored if `extra_pod_tolerations` is `None` or has length 0.

Returns

- **client** (*object*) – `dask.distributed.Client` connected to cluster
- **cluster** (*object*) – Pre-configured `dask_gateway.GatewayCluster`

See also:

`get_micro_cluster()` : A cluster with one-CPU workers

`get_standard_cluster()` : The default cluster specification

`get_big_cluster()` : A cluster with workers twice the size of the default

`get_giant_cluster()` : A cluster with workers four times the size of the default

`rhg_compute_tools.kubernetes.get_cluster(*args, **kwargs)`

All arguments are optional. If not provided, defaults will be used. To view defaults, instantiate a `dask_gateway.Gateway` object and call `gateway.cluster_options()`.

Parameters

- **name** (*str, optional*) – Name of worker image to use (e.g. `rhodium/worker:latest`). If `None` (default), default to worker specified in `template_path`.
- **tag** (*str, optional*) – Tag of the worker image to use. Cannot be used in combination with `name`, which should include a tag. If provided, overrides the tag of the image specified in `template_path`. If `None` (default), the full image specified in `name` or `template_path` is used.
- **extra_pip_packages** (*str, optional*) – Extra pip packages to install on worker. Packages are installed using `pip install extra_pip_packages`.
- **profile** (*One of ["micro", "standard", "big", "giant"]*) – Determines size of worker. CPUs assigned are slightly under 1, 2, 4, and 8, respectively. Memory assigned is slightly over 6, 12, 24, and 48 GB, respectively.
- **cpus** (*float, optional*) – Set the CPUs requested for your workers as defined by `profile`. Will raise error if >7.5 , because our 8-CPU nodes need ~ 5 vCPU for `kubernetes` pods. (NOTE 12/15/20: This is currently set to 1 by default to allow for mapping big workflows across inputs, see <https://github.com/dask/dask-gateway/issues/364>).
- **cred_name** (*str, optional*) – Name of Google Cloud credentials file to use, equivalent to providing `cred_path='/opt/gcsfuse_tokens/{}.json'.format(cred_name)`. May not use if `cred_path` is specified.
- **cred_path** (*str, optional*) – Path to Google Cloud credentials file to use. May not use if `cred_name` is specified.
- **env_items** (*dict, optional*) – A dictionary of env variable ‘name’-‘value’ pairs to append to the env variables included in `template_path`, e.g.

```
{
    'MY_ENV_VAR': 'some string',
}
```

- **extra_worker_labels** (*dict, optional*) – Dictionary of `kubernetes` labels to apply to pods. `None` (default) results in no additional labels besides those in the template, as well as `jupyter_user`, which is inferred from the `JUPYTERHUB_USER`, or, if not set, the server’s hostname.

- **extra_pod_tolerations** (*list of dict, optional*) – List of pod toleration dictionaries. For example, to match a node pool NoSchedule toleration, you might provide:

```
extra_pod_tolerations=[
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    },
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org/dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    }
]
```

- **keep_default_tolerations** (*bool, optional*) – Whether to append (default) or replace the default tolerations. Ignored if `extra_pod_tolerations` is `None` or has length 0.

Returns

- **client** (*object*) – `dask.distributed.Client` connected to cluster
- **cluster** (*object*) – Pre-configured `dask_gateway.GatewayCluster`

See also:

`get_micro_cluster()` : A cluster with one-CPU workers

`get_standard_cluster()` : The default cluster specification

`get_big_cluster()` : A cluster with workers twice the size of the default

`get_giant_cluster()` : A cluster with workers four times the size of the default

`rhg_compute_tools.kubernetes.get_giant_cluster(*args, **kwargs)`

Start a cluster with 4x the memory and CPU per worker relative to default

All arguments are optional. If not provided, defaults will be used. To view defaults, instantiate a `dask_gateway.Gateway` object and call `gateway.cluster_options()`.

Parameters

- **name** (*str, optional*) – Name of worker image to use (e.g. `rhodium/worker:latest`). If `None` (default), default to worker specified in `template_path`.
- **tag** (*str, optional*) – Tag of the worker image to use. Cannot be used in combination with `name`, which should include a tag. If provided, overrides the tag of the image specified in `template_path`. If `None` (default), the full image specified in `name` or `template_path` is used.
- **extra_pip_packages** (*str, optional*) – Extra pip packages to install on worker. Packages are installed using `pip install extra_pip_packages`.
- **profile** (*One of ["micro", "standard", "big", "giant"]*) – Determines size of worker. CPUs assigned are slightly under 1, 2, 4, and 8, respectively. Memory assigned is slightly over 6, 12, 24, and 48 GB, respectively.

- **cpus** (*float, optional*) – Set the CPUs requested for your workers as defined by profile. Will raise error if >7.5, because our 8-CPU nodes need ~.5 vCPU for kubernetes pods. (NOTE 12/15/20: This is currently set to 1 by default to allow for mapping big workflows across inputs, see <https://github.com/dask/dask-gateway/issues/364>).
- **cred_name** (*str, optional*) – Name of Google Cloud credentials file to use, equivalent to providing `cred_path='/opt/gcsfuse_tokens/{}.json'.format(cred_name)`. May not use if `cred_path` is specified.
- **cred_path** (*str, optional*) – Path to Google Cloud credentials file to use. May not use if `cred_name` is specified.
- **env_items** (*dict, optional*) – A dictionary of env variable ‘name’-‘value’ pairs to append to the env variables included in `template_path`, e.g.

```
{
    'MY_ENV_VAR': 'some string',
}
```

- **extra_worker_labels** (*dict, optional*) – Dictionary of kubernetes labels to apply to pods. None (default) results in no additional labels besides those in the template, as well as `jupyter_user`, which is inferred from the `JUPYTERHUB_USER`, or, if not set, the server’s hostname.
- **extra_pod_tolerations** (*list of dict, optional*) – List of pod toleration dictionaries. For example, to match a node pool NoSchedule toleration, you might provide:

```
extra_pod_tolerations=[
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    },
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org/dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    }
]
```

- **keep_default_tolerations** (*bool, optional*) – Whether to append (default) or replace the default tolerations. Ignored if `extra_pod_tolerations` is None or has length 0.

Returns

- **client** (*object*) – `dask.distributed.Client` connected to cluster
- **cluster** (*object*) – Pre-configured `dask_gateway.GatewayCluster`

See also:

`get_micro_cluster()` : A cluster with one-CPU workers

`get_standard_cluster()` : The default cluster specification

`get_big_cluster()` : A cluster with workers twice the size of the default

`get_giant_cluster()` : A cluster with workers four times the size of the default

`rhg_compute_tools.kubernetes.get_micro_cluster(*args, **kwargs)`

Start a cluster with a single CPU per worker

All arguments are optional. If not provided, defaults will be used. To view defaults, instantiate a `dask_gateway.Gateway` object and call `gateway.cluster_options()`.

Parameters

- **name** (*str, optional*) – Name of worker image to use (e.g. `rhodium/worker:latest`). If `None` (default), default to worker specified in `template_path`.
- **tag** (*str, optional*) – Tag of the worker image to use. Cannot be used in combination with `name`, which should include a tag. If provided, overrides the tag of the image specified in `template_path`. If `None` (default), the full image specified in `name` or `template_path` is used.
- **extra_pip_packages** (*str, optional*) – Extra pip packages to install on worker. Packages are installed using `pip install extra_pip_packages`.
- **profile** (*One of ["micro", "standard", "big", "giant"]*) – Determines size of worker. CPUs assigned are slightly under 1, 2, 4, and 8, respectively. Memory assigned is slightly over 6, 12, 24, and 48 GB, respectively.
- **cpus** (*float, optional*) – Set the CPUs requested for your workers as defined by `profile`. Will raise error if >7.5 , because our 8-CPU nodes need ~ 0.5 vCPU for `kubernetes` pods. (NOTE 12/15/20: This is currently set to 1 by default to allow for mapping big workflows across inputs, see <https://github.com/dask/dask-gateway/issues/364>).
- **cred_name** (*str, optional*) – Name of Google Cloud credentials file to use, equivalent to providing `cred_path='/opt/gcsfuse_tokens/{}.json'.format(cred_name)`. May not use if `cred_path` is specified.
- **cred_path** (*str, optional*) – Path to Google Cloud credentials file to use. May not use if `cred_name` is specified.
- **env_items** (*dict, optional*) – A dictionary of env variable ‘name’-‘value’ pairs to append to the env variables included in `template_path`, e.g.

```
{
    'MY_ENV_VAR': 'some string',
}
```

- **extra_worker_labels** (*dict, optional*) – Dictionary of `kubernetes` labels to apply to pods. `None` (default) results in no additional labels besides those in the template, as well as `jupyter_user`, which is inferred from the `JUPYTERHUB_USER`, or, if not set, the server’s hostname.
- **extra_pod_tolerations** (*list of dict, optional*) – List of pod toleration dictionaries. For example, to match a node pool `NoSchedule` toleration, you might provide:

```
extra_pod_tolerations=[
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    },
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
```

(continues on next page)

(continued from previous page)

```

        "operator": "Equal",
        "value": "worker-highcpu"
    }
]

```

- **keep_default_tolerations** (*bool, optional*) – Whether to append (default) or replace the default tolerations. Ignored if `extra_pod_tolerations` is `None` or has length 0.

Returns

- **client** (*object*) – `dask.distributed.Client` connected to cluster
- **cluster** (*object*) – Pre-configured `dask_gateway.GatewayCluster`

See also:

`get_micro_cluster()` : A cluster with one-CPU workers

`get_standard_cluster()` : The default cluster specification

`get_big_cluster()` : A cluster with workers twice the size of the default

`get_giant_cluster()` : A cluster with workers four times the size of the default

`rhg_compute_tools.kubernetes.get_standard_cluster(*args, **kwargs)`

Start a cluster with 1x the memory and CPU per worker relative to default

All arguments are optional. If not provided, defaults will be used. To view defaults, instantiate a `dask_gateway.Gateway` object and call `gateway.cluster_options()`.

Parameters

- **name** (*str, optional*) – Name of worker image to use (e.g. `rhodium/worker:latest`). If `None` (default), default to worker specified in `template_path`.
- **tag** (*str, optional*) – Tag of the worker image to use. Cannot be used in combination with `name`, which should include a tag. If provided, overrides the tag of the image specified in `template_path`. If `None` (default), the full image specified in `name` or `template_path` is used.
- **extra_pip_packages** (*str, optional*) – Extra pip packages to install on worker. Packages are installed using `pip install extra_pip_packages`.
- **profile** (*One of ["micro", "standard", "big", "giant"]*) – Determines size of worker. CPUs assigned are slightly under 1, 2, 4, and 8, respectively. Memory assigned is slightly over 6, 12, 24, and 48 GB, respectively.
- **cpus** (*float, optional*) – Set the CPUs requested for your workers as defined by `profile`. Will raise error if >7.5 , because our 8-CPU nodes need ~ 0.5 vCPU for `kubernetes` pods. (NOTE 12/15/20: This is currently set to 1 by default to allow for mapping big workflows across inputs, see <https://github.com/dask/dask-gateway/issues/364>).
- **cred_name** (*str, optional*) – Name of Google Cloud credentials file to use, equivalent to providing `cred_path='/opt/gcsfuse_tokens/{}.json'.format(cred_name)`. May not use if `cred_path` is specified.
- **cred_path** (*str, optional*) – Path to Google Cloud credentials file to use. May not use if `cred_name` is specified.

- **env_items** (*dict, optional*) – A dictionary of env variable ‘name’-‘value’ pairs to append to the env variables included in `template_path`, e.g.

```
{
    'MY_ENV_VAR': 'some string',
}
```

- **extra_worker_labels** (*dict, optional*) – Dictionary of kubernetes labels to apply to pods. None (default) results in no additional labels besides those in the template, as well as `jupyter_user`, which is inferred from the `JUPYTERHUB_USER`, or, if not set, the server’s hostname.
- **extra_pod_tolerations** (*list of dict, optional*) – List of pod toleration dictionaries. For example, to match a node pool NoSchedule toleration, you might provide:

```
extra_pod_tolerations=[
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org_dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    },
    {
        "effect": "NoSchedule",
        "key": "k8s.dask.org/dedicated",
        "operator": "Equal",
        "value": "worker-highcpu"
    }
]
```

- **keep_default_tolerations** (*bool, optional*) – Whether to append (default) or replace the default tolerations. Ignored if `extra_pod_tolerations` is None or has length 0.

Returns

- **client** (*object*) – `dask.distributed.Client` connected to cluster
- **cluster** (*object*) – Pre-configured `dask_gateway.GatewayCluster`

See also:

`get_micro_cluster()` : A cluster with one-CPU workers

`get_standard_cluster()` : The default cluster specification

`get_big_cluster()` : A cluster with workers twice the size of the default

`get_giant_cluster()` : A cluster with workers four times the size of the default

`rhg_compute_tools.kubernetes.traceback` (*fn*)

4.1.5 rhg_compute_tools.utils module

```
class rhg_compute_tools.utils.NumpyEncoder(*, skipkeys=False, ensure_ascii=True,
                                             check_circular=True, allow_nan=True,
                                             sort_keys=False, indent=None, separators=None, default=None)
```

Bases: `json.encoder.JSONEncoder`

Helper class for `json.dumps` to coerce numpy objects to native python

default (*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`rhg_compute_tools.utils.block_globals`

Decorator to prevent globals and undefined closures in functions and classes

Parameters

- **obj** (*function*) – Function to decorate. All globals not matching one of the allowed types will raise an `AssertionError`
- **allowed_types** (*type or tuple of types, optional*) – Types which are allowed as globals. By default, functions and modules are allowed. The full set of allowed types is drawn from the `types` module, and includes `FunctionType`, `ModuleType`, `MethodType`, `ClassType`, `BuiltinMethodType`, and `BuiltinFunctionType`.
- **include_defaults** (*bool, optional*) – If `allowed_types` is provided, setting `include_defaults` to `True` will append the default list of functions, modules, and methods to the user-passed list of allowed types. Default is `True`, in which case any user-passed elements will be added to the defaults described above. Setting to `False` will allow only the types passed in `allowed_types`.
- **whitelist** (*list of str, optional*) – Optional list of variable names to whitelist. If a list is provided, global variables will be compared to elements of this list based on their string names. Default (`None`) is no whitelist.

Examples

Wrap a function to block globals:

```
>>> my_data = 10

>>> @block_globals
... def add_5(data):
...     ''' can you spot the global? '''
...     a_number = 5
...     result = a_number + my_data
...     return result
Traceback (most recent call last):
...
TypeError: Illegal <class 'int'> global found in add_5: my_data
```

Wrapping a class will prevent globals from being used in all methods:

```
>>> @block_globals
... class MyClass:
...
...     @staticmethod
...     def add_5(data):
...         ''' can you spot the global? '''
...         a_number = 5
...         result = a_number + my_data
...         return result
Traceback (most recent call last):
...
TypeError: Illegal <class 'int'> global found in add_5: my_data
```

By default, functions and modules are allowed in the list of globals. You can modify this list with the `allowed_types` argument:

```
>>> result_formatter = 'my number is {}'
>>> @block_globals(allowed_types=str)
... def add_5(data):
...     ''' only allowed globals here! '''
...     a_number = 5
...     result = a_number + data
...     return result_formatter.format(result)
...
>>> add_5(3)
'my number is 8'
```

`block_globals` will also catch undefined references:

```
>>> @block_globals
... def get_mean(df):
...     return da.mean()
Traceback (most recent call last):
...
TypeError: Undefined global in get_mean: da
```

`rhg_compute_tools.utils.checkpoint` (*jobs, futures, job_name, log_dir='.', extra_pending=None, extra_errors=None, extra_others=None*)

checkpoint and save a job state to disk

`rhg_compute_tools.utils.collapse` (**args, **kwargs*)

Collapse positional and keyword arguments into an (args, kwargs) tuple

Intended for use with the `expand()` decorator

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns

- **args** (*tuple*) – Positional arguments tuple
- **kwargs** (*dict*) – Keyword argument dictionary

`rhg_compute_tools.utils.collapse_product` (**args, **kwargs*)

Parameters

- ***args** – Variable length list of iterables

- ****kwargs** – Keyword arguments, whose values must be iterables

Returns Generator with collapsed arguments

Return type iterator

See also:

Function() `py:func:collapse`

Examples

```
>>> @expand
... def my_func(a, b, exp=1):
...     return (a * b)**exp
...

>>> product_args = list(collapse_product(
...     [0, 1, 2],
...     [0.5, 2],
...     exp=[0, 1]))

>>> product_args
[(0, 0.5), {'exp': 0}],
[(0, 0.5), {'exp': 1}],
[(0, 2), {'exp': 0}],
[(0, 2), {'exp': 1}],
[(1, 0.5), {'exp': 0}],
[(1, 0.5), {'exp': 1}],
[(1, 2), {'exp': 0}],
[(1, 2), {'exp': 1}],
[(2, 0.5), {'exp': 0}],
[(2, 0.5), {'exp': 1}],
[(2, 2), {'exp': 0}],
[(2, 2), {'exp': 1}]]

>>> list(map(my_func, product_args))
[1.0, 0.0, 1, 0, 1.0, 0.5, 1, 2, 1.0, 1.0, 1, 4]
```

`rhg_compute_tools.utils.expand(func)`

Decorator to expand an (args, kwargs) tuple in function calls

Intended for use with the `collapse()` function

Parameters **func** (*function*) – Function to have arguments expanded. Func can have any number of positional and keyword arguments.

Returns **wrapped** – Wrapped version of func which accepts a single (args, kwargs) tuple.

Return type function

Examples

```
>>> @expand
... def my_func(a, b, exp=1):
...     return (a * b)**exp
...
```

(continues on next page)

(continued from previous page)

```
>>> my_func((2, 3), {})
6

>>> my_func((2, 3, 2), {})
36

>>> my_func((tuple([]), {'b': 4, 'exp': 2, 'a': 1}))
16
```

This function can be used in combination with the `collapse` helper function, which allows more natural parameter calls

```
>>> my_func(collapse(2, 3, exp=2))
36
```

These can then be paired to enable many parameterized function calls:

```
>>> func_calls = [collapse(a, a+1, exp=a) for a in range(5)]

>>> list(map(my_func, func_calls))
[1, 2, 36, 1728, 160000]
```

`rhg_compute_tools.utils.get_repo_state(repository_root: [<class 'str'>, None] = None) → dict`

Get a dictionary summarizing the current state of a repository.

Parameters `repository_root` (*str or None*) – Path to the root of the repository to document. If `None` (default), the current directory will be used, and will search parent directories for a git repository. If a string is passed, parent directories will not be searched - the directory must be a repository root which contains a `.git` directory.

Returns `repo_state` – Dictionary of repository information documenting the current state

Return type `dict`

class `rhg_compute_tools.utils.html` (*body*)
Bases: `object`

`rhg_compute_tools.utils.recover` (*job_name, log_dir=''*)
recover pending, errored, other jobs from a checkpoint

`rhg_compute_tools.utils.retry_with_timeout`
Execute `func` `n_tries` times, each time only allowing `retry_freq` seconds for the function to complete. There are two main cases where this could be useful:

1. You have a function that you know should execute quickly, but you may get occasional errors when running it simultaneously on a large number of workers. An example of this is massively parallelized I/O operations of `netcdfs` on GCS.
2. You have a function that may or may not take a long time, but you want to skip it if it takes too long.

There are two possible ways that this timeout function is implemented, each with pros and cons:

1. Using python's native `threading` module. If you are executing `func` outside of a dask worker, you likely will want this approach. It may be slightly faster and has the benefit of starting the timeout clock when the function starts executing (rather than when the function is *submitted* to a dask scheduler). **Note:** This approach will also work if calling `func` from a dask worker, but only if the cluster was set up such that `threads_per_worker=1`. Otherwise, this may cause issues if used from a dask worker.

2. Using `dask`. If you would like a dask worker to execute this function, you likely will want this approach. It can be executed from a dask worker regardless of the number of threads per worker (see above), but has the downside that the timeout clock begins once `func` is submitted, rather than when it begins executing.

Parameters

- **func** (*callable*) – The function you would like to execute with a timeout backoff.
- **retry_freq** (*float*) – The number of seconds to wait between successive retries of `func`.
- **n_tries** (*int*) – The number of retries to attempt before raising an error if none were successful
- **use_dask** (*bool*) – If true, will try to use the `dask`-based implementation (see description above). If no `Client` instance is present, will fall back to `use_dask=False`.

Returns

Return type The return value of `func`

Raises

- `dask.distributed.TimeoutError` : – If the function does not execute successfully in the specified `retry_freq`, after trying `n_tries` times.
- `ValueError` : – If `use_dask=True`, and a `Client` instance is present, but this function is executed from the client (rather than as a task submitted to a worker), you will get `ValueError("No workers found")`.

Examples

```
>>> import time
>>> @retry_with_timeout(retry_freq=.5, n_tries=1)
... def wait_func(timeout):
...     time.sleep(timeout)
>>> wait_func(.1)
>>> wait_func(1)
Traceback (most recent call last):
...
asyncio.exceptions.TimeoutError: Func did not complete successfully in allowed_
↳time/number of retries.
```

4.1.6 rhg_compute_tools.xarray module

`rhg_compute_tools.xarray.choose_along_axis(arr, axis=-1, replace=True, nchoices=1, p=None)`

Wrapper on `np.random.choice`, but along a single dimension within a larger array

Parameters

- **arr** (*np.array*) – Array with more than one dimension. Choices will be drawn from along the `axis` dimension.
- **axis** (*integer, optional*) – Dimension along which to draw samples
- **replace** (*bool, optional*) – Whether to sample with replacement. Passed to `np.random.choice()`. Default 1.

- **nchoices** (*int*, *optional*) – Number of samples to draw. Must be less than or equal to the number of valid options if `replace` is `False`. Default 1.
- **p** (*np.array*) – Array with the same shape as `arr` with weights for each choice. Each dimension is sampled independently, so weights will be normalized to 1 along the `axis` dimension.

Returns `sampled` – Array with the same shape as `arr` but with length `nchoices` along `axis` and with values chosen from the values of `arr` along dimension `axis` with weights `p`.

Return type `np.array`

Examples

Let's say we have an array with NaNs in it:

```
>>> arr = np.arange(40).reshape(4, 2, 5).astype(float)
>>> for i in range(4):
...     arr[i, :, i+1:] = np.nan
>>> arr
array([[ 0., nan, nan, nan, nan],
       [ 5., nan, nan, nan, nan],
       [10., 11., nan, nan, nan],
       [15., 16., nan, nan, nan],
       [20., 21., 22., nan, nan],
       [25., 26., 27., nan, nan],
       [30., 31., 32., 33., nan],
       [35., 36., 37., 38., nan]])
```

We can set weights such that we only select from non-nan values

```
>>> p = (~np.isnan(arr))
>>> p = p / p.sum(axis=2).reshape(4, 2, 1)
```

Now, sampling from this along the second dimension will draw from these values:

```
>>> np.random.seed(1)
>>> choose_along_axis(arr, 2, p=p, nchoices=10)
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.],
       [11., 11., 10., 11., 11., 11., 10., 10., 10., 11.],
       [15., 15., 16., 16., 16., 15., 16., 16., 15., 16.],
       [22., 22., 20., 22., 20., 21., 22., 20., 20., 20.],
       [25., 27., 25., 25., 26., 25., 26., 25., 26., 27.],
       [30., 31., 32., 31., 30., 32., 32., 32., 33., 32.],
       [38., 35., 35., 38., 36., 35., 38., 36., 38., 37.]])
```

See also:

`np.random.choice()` : 1-d version of this function

`rhg_compute_tools.xarray.choose_along_dim(da, dim, samples=1, expand=None, new_dim_name=None)`

Sample values from a DataArray along a dimension

Wraps `np.random.choice()` to sample a different random index (or set of indices) from along dimension `dim` for each combination of elements along the other dimensions. This is very different from block resampling - to block resample along a dimension simply choose a set of indices and draw these from the array using `xr.DataArray.sel()`.

Parameters

- **da** (*xr.DataArray*) – DataArray from which to sample values.
- **dim** (*str*) – Dimension along which to sample. Sampling will draw from elements along this dimension for all combinations of other dimensions.
- **samples** (*int*, *optional*) – Number of samples to take from the dimension *dim*. If greater than 1, *expand* is ignored (and set to True).
- **expand** (*bool*, *optional*) – Whether to expand the array along the sampled dimension.
- **new_dim_name** (*str*, *optional*) – Name for the new dimension. If not provided, will use *dim*.

Returns **sampler** – DataArray with sampled values chosen along dimension *dim*

Return type *xr.DataArray*

Examples

```
>>> da = xr.DataArray(
...     np.arange(40).reshape(4, 2, 5),
...     dims=['x', 'y', 'z'],
...     coords=[np.arange(4), np.arange(2), np.arange(5)],
... )

>>> da
<xarray.DataArray (x: 4, y: 2, z: 5)>
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],
       [[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]],
       [[20, 21, 22, 23, 24],
        [25, 26, 27, 28, 29]],
       [[30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39]]])
Coordinates:
  * x          (x) int64 0 1 2 3
  * y          (y) int64 0 1
  * z          (z) int64 0 1 2 3 4
```

We can take a random value along the 'z' dimension:

```
>>> np.random.seed(1)
>>> choose_along_dim(da, 'z')
<xarray.DataArray (x: 4, y: 2)>
array([[ 2,  8],
       [10, 16],
       [20, 25],
       [30, 36]])
Coordinates:
  * x          (x) int64 0 1 2 3
  * y          (y) int64 0 1
```

If you provide a *sample* argument greater than one (or set *expand=True*) the array will be expanded to a new dimension:

```

>>> np.random.seed(1)
>>> choose_along_dim(da, 'z', samples=3)
<xarray.DataArray (x: 4, y: 2, z: 3)>
array([[[ 2,  3,  0],
        [ 6,  5,  5]],
       [[10, 11, 11],
        [17, 17, 18]],
       [[21, 24, 20],
        [28, 27, 27]],
       [[30, 30, 34],
        [39, 36, 38]]])
Coordinates:
  * x          (x) int64 0 1 2 3
  * y          (y) int64 0 1
  * z          (z) int64 0 1 2

```

`rhg_compute_tools.xarray.dataarray_from_delayed(futures, dim=None, client=None, **client_kwargs)`

Returns a DataArray from a list of futures

Parameters

- **futures** (*list*) – list of `dask.delayed.Future` objects holding `xarray.DataArray` objects.
- **dim** (*str, optional*) – dimension along which to concat `xarray.DataArray`. Inferred by default.
- **client** (*object, optional*) – `dask.distributed.Client` to use in gathering metadata on futures. If not provided, client is inferred from context.
- **client_kwargs** (*optional*) – kwargs to pass to `client.map` and `client.gather` commands (e.g. `priority`)

Returns array – `xarray.DataArray` concatenated along `dim` with a `dask.array.Array` backend.

Return type `object`

Examples

Given a mapped xarray DataArray, pull the metadata into memory while leaving the data on the workers:

```

>>> import numpy as np, pandas as pd

>>> def build_arr(multiplier):
...     return multiplier * xr.DataArray(
...         np.arange(2), dims=['x'], coords=[['a', 'b']])
...

>>> client = dd.Client()
>>> fut = client.map(build_arr, range(3))
>>> da = dataarray_from_delayed(
...     fut,
...     dim=pd.Index(range(3), name='simulation'),
...     priority=1
... )
...

```

(continues on next page)

(continued from previous page)

```
>>> da
<xarray.DataArray ... (simulation: 3, x: 2)>
dask.array<...shape=(3, 2), dtype=int64, chunksize=(1, 2), chunktype=numpy.
↳ndarray>
Coordinates:
  * x                (x) <U1 'a' 'b'
  * simulation        (simulation) int64 0 1 2

>>> client.close()
```

`rhg_compute_tools.xarray.dataarrays_from_delayed(futures, client=None, **client_kwargs)`

Returns a list of xarray dataarrays from a list of futures of dataarrays

Parameters

- **futures** (*list*) – list of `dask.delayed.Future` objects holding `xarray.DataArray` objects.
- **client** (*object, optional*) – `dask.distributed.Client` to use in gathering metadata on futures. If not provided, client is inferred from context.
- **client_kwargs** (*optional*) – kwargs to pass to `client.map` and `client.gather` commands (e.g. `priority`)

Returns arrays – list of `xarray.DataArray` objects with `dask.array.Array` backends.

Return type list

Examples

Given a mapped xarray DataArray, pull the metadata into memory while leaving the data on the workers:

```
>>> import numpy as np

>>> def build_arr(multiplier):
...     return multiplier * xr.DataArray(
...         np.arange(2), dims=['x'], coords=[['a', 'b']])
...

>>> client = dd.Client()
>>> fut = client.map(build_arr, range(3))
>>> arrs = dataarrays_from_delayed(fut, priority=1)
>>> arrs[-1]
<xarray.DataArray ... (x: 2)>
dask.array<...shape=(2,), dtype=int64, chunksize=(2,), chunktype=numpy.ndarray>
Coordinates:
  * x                (x) <U1 'a' 'b'
```

This list of arrays can now be manipulated using normal xarray tools:

```
>>> xr.concat(arrs, dim='simulation')
<xarray.DataArray ... (simulation: 3, x: 2)>
dask.array<...shape=(3, 2), dtype=int64, chunksize=(1, 2), chunktype=numpy.
↳ndarray>
Coordinates:
```

(continues on next page)

(continued from previous page)

```
* x          (x) <U1 'a' 'b'
Dimensions without coordinates: simulation

>>> client.close()
```

`rhg_compute_tools.xarray.dataset_from_delayed(futures, dim=None, client=None, **client_kwargs)`

Returns an `xarray.Dataset` from a list of futures

Parameters

- **futures** (*list*) – list of `dask.delayed.Future` objects holding `xarray.Dataset` objects.
- **dim** (*str, optional*) – dimension along which to concat `xarray.Dataset`. Inferred by default.
- **client** (*object, optional*) – `dask.distributed.Client` to use in gathering metadata on futures. If not provided, client is inferred from context.
- **client_kwargs** (*optional*) – kwargs to pass to `client.map` and `client.gather` commands (e.g. `priority`)

Returns **dataset** – `xarray.Dataset` concatenated along `dim` with `dask.array.Array` backends for each variable.

Return type `object`

Examples

Given a mapped `xarray.Dataset`, pull the metadata into memory while leaving the data on the workers:

```
>>> import numpy as np, pandas as pd

>>> def build_ds(multiplier):
...     return multiplier * xr.Dataset({
...         'var1': xr.DataArray(
...             np.arange(2), dims=['x'], coords=[['a', 'b']])})
...

>>> client = dd.Client()
>>> fut = client.map(build_ds, range(3))
>>> ds = dataset_from_delayed(fut, dim=pd.Index(range(3), name='y'), priority=1)
>>> ds
<xarray.Dataset>
Dimensions:  (x: 2, y: 3)
Coordinates:
  * x        (x) <U1 'a' 'b'
  * y        (y) int64 0 1 2
Data variables:
    var1      (y, x) int64 dask.array<chunks=(1, 2), meta=np.ndarray>

>>> client.close()
```

`rhg_compute_tools.xarray.datasets_from_delayed(futures, client=None, **client_kwargs)`

Returns a list of `xarray` datasets from a list of futures of datasets

Parameters

- **futures** (*list*) – list of `dask.delayed.Future` objects holding `xarray.Dataset` objects.
- **client** (*object, optional*) – `dask.distributed.Client` to use in gathering metadata on futures. If not provided, client is inferred from context.
- **client_kwargs** (*optional*) – kwargs to pass to `client.map` and `client.gather` commands (e.g. `priority`)

Returns datasets – list of `xarray.Dataset` objects with `dask.array.Array` backends for each variable.

Return type list

Examples

Given a mapped `xarray.Dataset`, pull the metadata into memory while leaving the data on the workers:

```
>>> import numpy as np

>>> def build_ds(multiplier):
...     return multiplier * xr.Dataset({
...         'var1': xr.DataArray(
...             np.arange(2), dims=['x'], coords=[['a', 'b']])})
...

>>> client = dd.Client()
>>> fut = client.map(build_ds, range(3))
>>> arrs = datasets_from_delayed(fut, priority=1)
>>> arrs[-1]
<xarray.Dataset>
Dimensions:  (x: 2)
Coordinates:
  * x        (x) <U1 'a' 'b'
Data variables:
  var1      (x) int64 dask.array<chunksize=(2,), meta=np.ndarray>
```

This list of arrays can now be manipulated using normal `xarray` tools:

```
>>> xr.concat(arrs, dim='y')
<xarray.Dataset>
Dimensions:  (x: 2, y: 3)
Coordinates:
  * x        (x) <U1 'a' 'b'
Dimensions without coordinates: y
Data variables:
  var1      (y, x) int64 dask.array<chunksize=(1, 2), meta=np.ndarray>

>>> client.close()
```

`rhg_compute_tools.xarray.document_dataset` (*ds: xarray.core.dataset.Dataset, repository_root: [class 'str'], None] = None, tz: str = 'UTC', inplace: bool = True*) → `xarray.core.dataset.Dataset`

Add repository state and timestamp to dataset attrs

Parameters

- **ds** (*xr.Dataset*) – Dataset to document

- **repository_root** (*str* or *None*, *optional*) – Path to the root of the repository to document. If *None* (default), the current directory will be used, and will search parent directories for a git repository. If a string is passed, parent directories will not be searched - the directory must be a repository root which contains a `.git` directory.
- **tz** (*str*, *optional*) – time zone string parseable by `datetime.datetime` (e.g. “US/Pacific”). Default “UTC”.
- **inplace** (*bool*, *optional*) – Whether to update the dataset’s attributes in place (default) or to return a copy of the dataset.

Returns **ds** – Dataset with updated attribute information. A dataset is returned regardless of arguments - the `inplace` argument determines whether the returned dataset will be a shallow copy or the original object (default).

Return type `xr.Dataset`

class `rhg_compute_tools.xarray.random(xarray_obj)`

Bases: `object`

choice (*dim*, *samples=1*, *expand=None*, *new_dim_name=None*)

Sample values from a `DataArray` along a dimension

Wraps `np.random.choice()` to sample a different random index (or set of indices) from along dimension `dim` for each combination of elements along the other dimensions. This is very different from block resampling - to block resample along a dimension simply choose a set of indices and draw these from the array using `xr.DataArray.sel()`.

Parameters

- **da** (*xr.DataArray*) – `DataArray` from which to sample values.
- **dim** (*str*) – Dimension along which to sample. Sampling will draw from elements along this dimension for all combinations of other dimensions.
- **samples** (*int*, *optional*) – Number of samples to take from the dimension `dim`. If greater than 1, `expand` is ignored (and set to `True`).
- **expand** (*bool*, *optional*) – Whether to expand the array along the sampled dimension.
- **new_dim_name** (*str*, *optional*) – Name for the new dimension. If not provided, will use `dim`.

Returns **sampled** – `DataArray` with sampled values chosen along dimension `dim`

Return type `xr.DataArray`

Examples

```
>>> da = xr.DataArray(
...     np.arange(40).reshape(4, 2, 5),
...     dims=['x', 'y', 'z'],
...     coords=[np.arange(4), np.arange(2), np.arange(5)],
... )

>>> da
<xarray.DataArray (x: 4, y: 2, z: 5)>
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],
```

(continues on next page)

(continued from previous page)

```

        [[10, 11, 12, 13, 14],
         [15, 16, 17, 18, 19]],
        [[20, 21, 22, 23, 24],
         [25, 26, 27, 28, 29]],
        [[30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]]])
Coordinates:
* x          (x) int64 0 1 2 3
* y          (y) int64 0 1
* z          (z) int64 0 1 2 3 4

```

We can take a random value along the 'z' dimension:

```

>>> np.random.seed(1)
>>> choose_along_dim(da, 'z')
<xarray.DataArray (x: 4, y: 2)>
array([[ 2,  8],
       [10, 16],
       [20, 25],
       [30, 36]])
Coordinates:
* x          (x) int64 0 1 2 3
* y          (y) int64 0 1

```

If you provide a sample argument greater than one (or set `expand=True`) the array will be expanded to a new dimension:

```

>>> np.random.seed(1)
>>> choose_along_dim(da, 'z', samples=3)
<xarray.DataArray (x: 4, y: 2, z: 3)>
array([[[ 2,  3,  0],
        [ 6,  5,  5]],
       [[10, 11, 11],
        [17, 17, 18]],
       [[21, 24, 20],
        [28, 27, 27]],
       [[30, 30, 34],
        [39, 36, 38]]])
Coordinates:
* x          (x) int64 0 1 2 3
* y          (y) int64 0 1
* z          (z) int64 0 1 2

```

4.1.7 Module contents

Top-level package for RHG Compute Tools.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/RhodiumGroup/rhg_compute_tools/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

5.1.4 Write Documentation

RHG Compute Tools could always use more documentation, whether as part of the official RHG Compute Tools docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/RhodiumGroup/rhg_compute_tools/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *rhg_compute_tools* for local development.

1. Fork the *rhg_compute_tools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/rhg_compute_tools.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv rhg_compute_tools
$ cd rhg_compute_tools/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 rhg_compute_tools tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. Check https://github.com/RhodiumGroup/rhg_compute_tools/actions and make sure that the tests pass.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_rhg_compute_tools
```


6.1 Current version (unreleased)

- Add *utils.get_repo_state* and *xarray.document_dataset* functions
- Drop explicit testing of *dask.gateway rpy2* functionality
- Bugfix in sphinx docs
- Fix to avoid internal calls to GCS API requiring unnecessary *storage.bucket.get* permissions.

6.2 v1.2.1

Bug fixes: * raise error on gsutil nonzero status in *rhg_compute_tools.gcs.cp* (PR #105)

6.3 v1.2

New features: * Adds google storage directory marker utilities and *rc tools gcs mkdirs* command line app

6.4 v1.1.4

- Add *dask_kwargs* to the *rhg_compute_tools.xarray* functions

6.5 v1.1.3

- Add *retry_with_timeout* to *rhg_compute_tools.utils.py*

6.6 v1.1.2

- Drop `matplotlib.font_manager._rebuild()` call in `design.__init__` - no longer supported (GH #96)

6.7 v1.1.1

- Refactor `datasets_from_delayed` to speed up

6.8 v1.1

- Add `gcs.ls` function

6.9 v1.0.1

- Fix `tag` kwarg in `get_cluster`

6.10 v1.0.0

- Make the gsutil API consistent, so that we have `cp`, `sync` and `rm`, each of which accept the same args and kwargs (GH #69)
- Swap `bumpversion` for `setuptools_scm` to handle versioning (GH #78)
- Cast `coordinates` to `dict` before gathering in `rhg_compute_tools.xarray.dataarrays_from_delayed` and `rhg_compute_tools.xarray.datasets_from_delayed`. This avoids a mysterious memory explosion on the local machine. Also add name in the metadata used by those functions so that the name of each dataarray or Variable is preserved. (GH #83)
- Use `dask-gateway` when available when creating a cluster in `rhg_compute_tools.kubernetes`. Add some tests using a local gateway cluster. TODO: More tests.
- Add `tag` kwarg to `rhg_compute_tools.kubernetes.get_cluster` function (PR #87)

6.11 v0.2.2

- ?

6.12 v0.2.1

- Add remote scheduler deployment (part of `dask_kubernetes 0.10`)
- Remove extraneous `GCSFUSE_TOKENS` env var no longer used in new worker images
- Set library thread limits based on how many cpus are available for a single dask thread

- Change formatting of the extra `env_items` passed to `get_cluster` to be a list rather than a list of dict-like name/value pairs

6.13 v0.2.0

- Add CLI tools ([GH #37](#)). See `rctools gcs repdirstruc --help` to start
- Add new function `rhg_compute_tools.gcs.replicate_directory_structure_on_gcs` to copy directory trees into GCS. Users can authenticate with `cred_file` or with default google credentials ([GH #51](#))
- Fixes to docstrings and metadata ([GH #43](#)) ([GH #45](#))
- Add new function `rhg_compute_tools.gcs.rm` to remove files/directories on GCS using the `google.cloud.storage` API
- Store one additional environment variable when passing `cred_path` to `rhg_compute_tools.kubernetes.get_cluster` so that the `google.cloud.storage` API will be authenticated in addition to `gsutil`

6.14 v0.1.8

- Deployment fixes

6.15 v0.1.7

- Design tools: use RHG & CIL colors & styles
- Plotting helpers: generate `cmaps` with consistent colors & norms, and apply a colorbar to `geopandas` plots with nonlinear norms
- Autoscaling fix for `kubecenter`: switch to `dask_kubernetes.KubeCluster` to allow use of recent bug fixes

6.16 v0.1.6

- Add `rhg_compute_tools.gcs.cp_gcs` and `rhg_compute_tools.gcs.sync_gcs` utilities

6.17 v0.1.5

- need to figure out how to use this `rever` thing

6.18 v0.1.4

- Bug fix again in `rhg_compute_tools.kubernetes.get_worker`

6.19 v0.1.3

- Bug fix in `rhg_compute_tools.kubernetes.get_worker`

6.20 v0.1.2

- Add `xarray` from delayed methods in `rhg_compute_tools.xarray` ([GH #s12](#))
- `rhg_compute_tools.gcs.cp_to_gcs` now calls `gsutil` in a subprocess instead of `google.storage` operations. This dramatically improves performance when transferring large numbers of small files ([GH #s11](#))
- Additional cluster creation helpers ([GH #s3](#))

6.21 v0.1.1

- New google compute helpers (see `rhg_compute_tools.gcs.cp_to_gcs`, `rhg_compute_tools.gcs.get_bucket`)
- New cluster creation helper (see `rhg_compute_tools.kubernetes.get_worker`)
- Dask client.map helpers (see `rhg_compute_tools.utils` submodule)

6.22 v0.1.0

- First release on PyPI.

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

r

- `rhg_compute_tools`, [34](#)
- `rhg_compute_tools.design`, [10](#)
- `rhg_compute_tools.design.colors`, [9](#)
- `rhg_compute_tools.design.plotting`, [9](#)
- `rhg_compute_tools.gcs`, [12](#)
- `rhg_compute_tools.io_tools`, [12](#)
- `rhg_compute_tools.io_tools.readers`, [11](#)
- `rhg_compute_tools.io_tools.writers`, [12](#)
- `rhg_compute_tools.kubernetes`, [14](#)
- `rhg_compute_tools.utils`, [21](#)
- `rhg_compute_tools.xarray`, [26](#)

A

`add_colorbar()` (in module *rhg_compute_tools.design*), 10
`add_colorbar()` (in module *rhg_compute_tools.design.plotting*), 9
`authenticated_client()` (in module *rhg_compute_tools.gcs*), 12

B

`block_globals` (in module *rhg_compute_tools.utils*), 22

C

`checkpoint()` (in module *rhg_compute_tools.utils*), 23
`choice()` (*rhg_compute_tools.xarray.random* method), 33
`choose_along_axis()` (in module *rhg_compute_tools.xarray*), 26
`choose_along_dim()` (in module *rhg_compute_tools.xarray*), 27
`collapse()` (in module *rhg_compute_tools.utils*), 23
`collapse_product()` (in module *rhg_compute_tools.utils*), 23
`cp()` (in module *rhg_compute_tools.gcs*), 12
`create_directories_under_blob()` (in module *rhg_compute_tools.gcs*), 13
`create_directory_markers()` (in module *rhg_compute_tools.gcs*), 13

D

`dataarray_from_delayed()` (in module *rhg_compute_tools.xarray*), 29
`dataarrays_from_delayed()` (in module *rhg_compute_tools.xarray*), 30
`dataset_from_delayed()` (in module *rhg_compute_tools.xarray*), 31
`datasets_from_delayed()` (in module *rhg_compute_tools.xarray*), 31

`default()` (*rhg_compute_tools.utils.NumpyEncoder* method), 22

`document_dataset()` (in module *rhg_compute_tools.io_tools.writers*), 12
`document_dataset()` (in module *rhg_compute_tools.xarray*), 32

E

`expand()` (in module *rhg_compute_tools.utils*), 24

G

`get_big_cluster()` (in module *rhg_compute_tools.kubernetes*), 14
`get_bucket()` (in module *rhg_compute_tools.gcs*), 13
`get_cluster()` (in module *rhg_compute_tools.kubernetes*), 16
`get_color_scheme()` (in module *rhg_compute_tools.design*), 10
`get_color_scheme()` (in module *rhg_compute_tools.design.plotting*), 9
`get_fs()` (in module *rhg_compute_tools.io_tools.readers*), 11
`get_giant_cluster()` (in module *rhg_compute_tools.kubernetes*), 17
`get_maximal_chunks_encoding()` (in module *rhg_compute_tools.io_tools.writers*), 12
`get_micro_cluster()` (in module *rhg_compute_tools.kubernetes*), 18
`get_repo_state()` (in module *rhg_compute_tools.utils*), 25
`get_standard_cluster()` (in module *rhg_compute_tools.kubernetes*), 20

H

`html` (class in *rhg_compute_tools.utils*), 25

L

`load_netcdf()` (in module *rhg_compute_tools.io_tools.readers*), 11

`ls()` (in module `rhg_compute_tools.gcs`), 13

N

`NumpyEncoder` (class in `rhg_compute_tools.utils`), 21

R

`random` (class in `rhg_compute_tools.xarray`), 33

`read_csv()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_csvv_response()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_dataframe()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_dataset()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_netcdf()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_parquet()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_rasterio()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_shapefile()` (in module `rhg_compute_tools.io_tools.readers`), 11

`read_zarr()` (in module `rhg_compute_tools.io_tools.readers`), 11

`recover()` (in module `rhg_compute_tools.utils`), 25

`replicate_directory_structure_on_gcs()`
(in module `rhg_compute_tools.gcs`), 13

`report_if_not_found()` (in module `rhg_compute_tools.io_tools.readers`), 11

`retry_with_timeout` (in module `rhg_compute_tools.utils`), 25

`rhg_compute_tools` (module), 34

`rhg_compute_tools.design` (module), 10

`rhg_compute_tools.design.colors` (module), 9

`rhg_compute_tools.design.plotting` (module), 9

`rhg_compute_tools.gcs` (module), 12

`rhg_compute_tools.io_tools` (module), 12

`rhg_compute_tools.io_tools.readers` (module), 11

`rhg_compute_tools.io_tools.writers` (module), 12

`rhg_compute_tools.kubernetes` (module), 14

`rhg_compute_tools.utils` (module), 21

`rhg_compute_tools.xarray` (module), 26

`rm()` (in module `rhg_compute_tools.gcs`), 14

S

`sync()` (in module `rhg_compute_tools.gcs`), 14

T

`traceback()` (in module `rhg_compute_tools.kubernetes`), 21

W

`write_netcdf()` (in module `rhg_compute_tools.io_tools.writers`), 12

`write_zarr()` (in module `rhg_compute_tools.io_tools.writers`), 12