tucluster Documentation

Release 0.1.0

James Ramm

Aug 03, 2017

Contents

TuCluster is a HTTP API (i.e. a web service) for queueing, executing and managing flood models in the cloud or a local cluster. It is designed to work with Tuflow.

Contents:

CHAPTER 1

Installation

Getting Prepared

TuCluster is a distributed/cloud service for running flood models. A typical setup will have at least 2 hardware servers: One server on which the TuCluster API runs and one or more 'worker nodes' which are responsible for running Anuga/Tuflow and receiving tasks. (For local development, you will typically install everything onto your single development computer.)

Setting Up The Main Server

You will need to install **'Redis<https://redis.io/>'_** and **'MongoDB<https://www.mongodb.com/>'_**. Redis is a message queue which will pass instructions from your main server to each worker node. MongoDB is a NoSQL database which will store metadata about your models.

After installation of Redis and MongoDB, you can go ahead and install TuCluster:

To install the latest stable release, run this command in your terminal:

```
$ pip install tucluster
```

If you don't have pip installed, this Python installation guide can guide you through the process.

Note that TuCluster is only compatible with Python 3.5 and above. We recommend installing into a virtualenv.

Setting up the Workers

For each worker we will install:

- TuCluster (tucluster contains both the main server API and the celery worker application. We will use different commands to start running the different apps)
- ANUGA

You can install Tucluster as above (if your worker node and main server are the same, there is no need to install twice!)

ANUGA Setup

In order to run ANUGA without conflicts with Tucluster library, TuCluster expects that ANUGA is installed into a conda environment called 'anuga'.

1. Install Miniconda on each of your worker nodes. (https://conda.io/miniconda.html)
2. **Create a new environment and install the dependencies:** ` conda create -n anuga python=2 pip nomkl nose numpy scipy matplotlib netcdf4 source activate anuga conda install -c pingucarsti gdal export GDAL_DATA=`gdal-config --datadir` `
3. Clone Anuga to a suitable location on the worker: `git clone https://github.com/GeoscienceAustralia/anuga_core`
4. **cd into the anuga directory and install:** ` python setup.py build sudo python setup.py install `

It should be noted that all the nodes in the cluster (workers and server) should have a common file system in which the modelling data and results will be placed.

CHAPTER 2

Setting Up

Configuration

TuCluster comes with a default configuration suitable for development, but for running in production you will want to create a configuration file. This is a simple JSON file. You should create an environment variable called `TUCLUSTER_CONFIG` which points to its' location.

Tucluster will override the default configuration with the contents of this file.

Example config file:

```
{
    "MONGODB": {
        "db": "tucluster",
        "host": "127.0.0.1",
        "port": 27017
    },
    "MODEL_DATA_DIR": "/path/to/data/dir",
    "TUFLOW_PATH": "tuflow",
    "ANUGA_ENV": "anuga"
}
```

Here is an explanation of the keys in the above config:

MONGODB This defines the database connection details. When you installed mongodb, you should've created a database; in the above configuration we have called this "tucluster". If you secured your database, you can add a `"username"` and `"password"` attribute after the `"port"`

MODEL_DATA_DIR This is path to the root directory to where all model data should be put. This is the user uploaded input data and the result data created by e.g. tuflow. Therefore, you should ensure there is enough space to support your modelling needs. All the nodes in your cluster need to be able to access this path, so you will typically use some form of distributed file system. You should choose a file system with the lowest possible latency and that is easily scaled as your model output grows.

Once Tucluster is running, you should *never* interact directly with this folder. TuCluster will manage the storage, upload and download of files. Manually adding/removing files could cause exceptions

in TuClusters' execution.

TUFLOW_PATH Path to the Tuflow executable to use on each worker node. This implies that it should be the same on each node. If Tuflow is available globally on the system PATH, just enter the executable name

ANUGA_ENV Name of the conda environment in which ANUGA is installed.

Running TuCluster

You should use a production-ready web server such as GUnicorn to run tucluster. Running with GUnicorn is easy:

```
gunicorn tucluster.app
```

This will run tucluster on port 8000. You would normally configure a proxy such as NGinx to allow external requests.

You will then need to run the tucluster celery app on each worker node:

celery -A qflow worker -l info

Run the above command on each server which you wish to execute tuflow models (remember; you will need to install tucluster on each of these nodes aswell!).

You are now ready to start interacting with TuCluster

Note on Using ANUGA

Running python scripts on the server is potentially dangerous and you should ensure that your webserver runs as a non-root user and restrict its permissions. Essentially, the only place it needs to be able to write files is the directory you set as the `MODEL_DATA_DIR` in your configuration file.

Developers of ANUGA scripts should be aware of these restrictions and should set the `datadir` appropriately. The best advice here is to set it relative to the script file. E.g:

```
domain.set_datadir(os.path.dirname(__file__), 'results')
```

ANUGA scripts should also be carefully developed so they do not hang. E.g. causing the script to display a matplotlib figure or other GUI elements will prevent the script from terminating until the window has been manually closed. You should not do this! Any figures should be output directly to file.

CHAPTER 3

Usage

You interact with Tucluster via some form of web client. This could be a user-facing website you create yourself (we are in the process of creating a client website for using TuCluster) or via a command line tool such as cUrl or HTTPie. I reccomend the latter for command line usage.

Typical Workflow

When using TuCluster, you will typically follow a workflow similar to the following:

- Prepare an input zip folder. This is all the modelling input data and ANUGA scripts (or Tuflow control files). Note that

the scripts/control files should be in the root zip archive.

- Upload the zip archive to TuCluster to create a new Model.
- You may wish to update the description or name of your newly created model, or explore other models on the system. You can view the input data folder structure and download individual files.
- Start a new modelling task. In Tucluster, this is called a Model Run. Each model run has a parent Model, a modelling engine and an entrypoint to be used. An entrypoint is an ANUGA script or control file. The modelling engine is either `'anuga'` or `'tuflow'`. The modelling task will then be added to the queue and executed in the background by one of the worker nodes. You can create as many modelling tasks as you like - they will simply be queued up and executed when a worker is available.
- Check the status of your modelling task. Each model run has a task id which can be used to check the status of the task and get the result location once it has successfully completed.
- Explore the results. Once the modelling task has completed, you can explore the result folder, aswell as the output of the tuflow checks and logs (or any other output you create from ANUGA). You can find specific files and download them to your local computer.
- Explore historic models and runs. The metadata for all Models and model runs are tagged with dates and users so you can easily get a record of data provenance or build up an overview of what models you have available. You may mark a model run within a model as the `baseline` run for a given model.

We will now review each of the above points in detail and explain how to interact with TuCluster to achieve them.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/JamesRamm/tucluster/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.

Fix Bugs

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

Implement Features

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

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/JamesRamm/tucluster/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 :)

Get Started!

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

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

```
$ git clone git@github.com:your_name_here/tucluster.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 tucluster
$ cd tucluster/
$ 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 tucluster tests
$ python setup.py test or py.test
$ 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.

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. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/JamesRamm/tucluster/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_tucluster
```

CHAPTER 5

Credits

Development Lead

- James Ramm <jamessramm@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.1.0 (2017-08-03)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search