# Lightflow Documentation

***Release 1.7.3***

**Software Engineering Australian Synchrotron Software Group**

**Nov 20, 2017**

# Contents

Lightflow is a Python 3.5+ library and command-line tool for executing workflows, composed of individual tasks, in a distributed fashion. It is based on Celery and provides task dependencies, data exchange between tasks and an intuitive description of workflows.

CHAPTER 1

# User's Guide

## 1.1 Installation

One of the key goals when developing Lightflow was to keep the infrastructure dependencies as small as possible. Lightflow does not require special file systems, job scheduler systems or special hardware. It runs on most Linux distributions and is known to work on MacOSX as well as on the Ubuntu subsystem of Windows 10. Apart from Python 3.5+, the only dependencies of Lightflow are a running redis and MongoDB database.

### 1.1.1 Python

Lightflow requires Python 3.5 or higher. It has been developed and tested with both a native Python installation as well as Miniconda/Anaconda.

Lightflow's main Python dependencies are:

- Celery - for queuing and managing jobs and running workers
- NetworkX - for building and interrogating the directed acyclic graphs of a workflow
- cloudpickle - for exchanging data between tasks running on distributed hosts
- Click - for the command line client
- ruamel.yaml - for reading the configuration file

These dependencies are installed during the installation of Lightflow automatically.

### 1.1.2 redis

Redis is an in-memory key-value database and is required by Lightflow as a communication broker between tasks. It is also used as the default broker for the Celery queuing system, but could be replaced with any other supported Celery broker.

You can either download redis from the offical redis website or install it via the package manager of your distribution. By default, the redis server runs on `localhost` and port `6379`. The *Quickstart* as well as the *Tutorial* assume you are running redis using these defaults.

### 1.1.3 MongoDB

MongoDB is a popular document-oriented database and is used by Lightflow for storing data that should persist during a workflow run.

You can either download MongoDB from the official MongoDB website or install it via the package manager of your distribution:

- RedHat
- Debian
- Ubuntu

By default, MongoDB runs on `localhost` and port `27017`. The *Quickstart* as well as the *Tutorial* assume you are running MongoDB using these defaults.

### 1.1.4 Lightflow

After having redis and MongoDB running, installing Lightflow is a breeze. It is available from PyPI and can be installed with:

```
pip install lightflow
```

This will install the Lightflow libraries, command line client and example workflows.

## 1.2 Quickstart

You can't wait to start using Lightflow or have no time to follow the tutorial? No problem, just spend a few minutes with this quickstart guide and you are on your way to using Lightflow.

This quickstart guide assumes that you have a redis database running on `localhost` and port `6379`, a MongoDB database running on `localhost` and port `27017` as well as Lightflow installed on your system. If you haven't installed the database systems and Lightflow yet, no problem, just follow the *Installation* guide.

### 1.2.1 Configuration and examples

Create an empty directory in your preferred location. We will use this directory in the following to store the configuration file and the example workflows. Lightflow has no restrictions on where this directory should be located and what its name should be.

The first step is to create the global configuration file for Lightflow. This file contains, among other settings, the connection information for redis and MongoDB. The quickest and easiest way to generate a default configuration file is to use the Lightflow command line interface. Make sure you are located in the directory you created earlier and enter:

```
$ lightflow config default .
```

This will create a configuration file called `lightflow.cfg` containing a default configuration. If you were running redis and MongoDB on different hosts than `localhost` or the default port, edit the appropriate settings in the configuration file. You can find more information about the configuration file in the section Configuration.

Lightflow ships with a number of examples that demonstrate various features of the system. We will copy these examples into a subfolder called `examples` inside your current directory. This will allow you to modify the examples as you see fit or use them as a starting point for your own workflows. The command line interface offers a quick and easy way to copy the examples:

```
$ lightflow config examples .
```

Now you will find a subfolder `examples` in your directory containing all example workflows. If you like, you can list all available example workflows together with a short description, Make sure you are located in the folder containing the configuration file, then enter:

```
$ lightflow workflow list
```

### 1.2.2 Start the workers

Lightflow uses a worker based scheme. This means a workflow adds jobs onto a central queue from which a number of workers consume jobs and execute them. In order for Lightflow to run a workflow, it needs at least one running worker (obviously). You can start a worker with:

```
$ lightflow worker start
```

This will start a worker, which then waits for the first job to be added to the queue. You can start as many workers as you like, but for the quickstart guide one worker is enough.

---

**A recommended setup for multiple workers**

What is special about Lightflow, in comparison with other workflow systems, is that it also uses workers for running the workflow itself. This means, there is no central daemon and thus no single point of failure. Lightflow uses three queues for running a workflow. Two queues, labelled `workflow` and `dag`, for managing the workflows and one queue, labelled `task`, for executing the individual tasks of a workflow. A typical setup of workers would consist of one worker dealing with workflow related jobs, thus consuming jobs only from the `workflow` and `dag` queues, and one or more workers executing the actual tasks.

You can use the `-q` argument in the command line interface in order to restrict the queues a worker consumes jobs from. For the recommended setup discussed above you would start one worker with:

```
$ lightflow worker start -q workflow,dag
```

and at least one more worker with:

```
$ lightflow worker start -q task
```

---

### 1.2.3 Run a workflow

With at least one worker running, we are ready to run our first workflow. You can pick any example workflow you like and run it. In the following we will run the most basic of all workflows, the `simple` workflow. You might need a second terminal in order to run the workflow as the first one is occupied running our worker. In your second terminal enter:

```
$ lightflow workflow start simple
```

This will send the workflow `simple` to the queue. Our worker will pick up the workflow and run it. The default logging level is very verbose so you will see the worker print out a lot of information as it executes the workflow.

### 1.2.4 Where to go from here

Congratulations, you have finished the quickstart guide. A good place to continue is to have a look at the documented example workflows. They are a great starting point for exploring the features of Lightflow. Alternatively, head over to the tutorial section for a more structured introduction to Lightflow.

## 1.3 Tutorial

Welcome to the Lightflow tutorial! This tutorial will guide you step by step through the development of a workflow. The emphasis is on showing you how to model and implement typical workflow elements with Lightflow and to demonstrate the features Lightflow has to offer. Therefore, the workflow we are going to create does nothing particularly sophisticated. So don't expect too much. We will push numbers around and perform some basic math operations on them.

Let's go! While we recommend that you follow the tutorial steps in order, you don't have to. Each tutorial step introduces a specific concept or feature of Lightflow, so feel free to jump directly to the step that interests you most.

### 1.3.1 Step 0: Setup

In this step we will set up the environment for the tutorial and create an empty workflow. We assume that you followed the *Installation* guide and have a redis database running on `localhost` and port `6379`, a MongoDB database running on `localhost` and port `27017` as well as Lightflow installed on your system.

To test whether you have installed Lightflow correctly, enter the following into a terminal:

```
$ lightflow
```

This calls the command line interface of Lightflow and will print the available commands and options.

#### Configuration file

Start by creating an empty directory with your preferred name (e.g. `lightflow_tutorial`) in a location of your choice (e.g. your home directory). This will be our working directory for the tutorial containing the lightflow configuration file and our workflow file. Lightflow has no restrictions on where this directory should be located and what it is called.

Next, we will create a configuration file. Lightflow uses the configuration file for storing settings such as the connection information for redis and MongoDB, and the location of your workflow files. To make things easier, we equipped the command line interface with a command to generate a default configuration file for you. Make sure you are located in the directory you created earlier, then enter:

```
$ lightflow config default .
```

This creates a configuration file called `lightflow.cfg` containing the default settings into the current directory.

Let's have a look into the configuration file. Open `lightflow.cfg` with your editor of choice. The configuration file uses the YAML format and is broken up into several sections.

**Non-default redis or MongoDB**

If you are running redis on a different host or port from the default mentioned above, change the host and port settings in the `celery` as well as `signal` sections in the configuration file. If your MongoDB configuration deviates from the default, edit the host and port fields in the `store` section.

We will focus on the first field labelled `workflows`. This field contains a list of paths where Lightflow should look for workflow files. The paths can either be relative to the configuration file or absolute paths. By default, Lightflow expects to find workflow files in a sub-directory called `examples`, located in the same directory as your configuration file. However, we would like our tutorial workflow file to live in its own directory called `tutorial`. Therefore, edit the configuration file by changing `examples` to `tutorial`:

```
workflows:
  - ./tutorial
```

Save the file and exit your editor.

#### Tutorial folder

Before we can move on we have to create the `tutorial` folder for our tutorial workflow file of course. In the same directory as your configuration file, create a sub-directory called `tutorial`:

```
$ mkdir tutorial
```

Now you are ready to write your first workflow! Head over to *Step 1: A simple workflow* in our tutorial and learn how to write your first workflow.

### 1.3.2 Step 1: A simple workflow

In this section of the tutorial we will write our first workflow. It will consist of two tasks that are executed in order. Each task will print a message so you can track the execution of the tasks. At the end of this section you will have learned how to create tasks, arrange their execution order and run a workflow.

#### Workflow file

In Lightflow, workflows are defined using Python. This means you don't have to learn another language and you can use your favorite Python libraries and modules. Typically you would have a single Python file describing the entire workflow, but for complex workflows you can, of course, split the workflow definition into multiple files. For this tutorial, we will only have a single workflow file.

Change into the `tutorial` directory and create an empty file called `tutorial01.py`. This file will contain the workflow for this step of the tutorial. Your directory structure should look like this:

```
/lightflow_tutorial
    lightflow.cfg
    /tutorial
        tutorial01.py
```

### Create two tasks

Let's get started with our workflow. First, we will create the two tasks for our small workflow. Open the workflow file you just created with your editor of choice. At the top of the file import the PythonTask class:

```
from lightflow.tasks import PythonTask
```

Lightflow is shipped with two task classes: the `PythonTask` and the `BashTask`. The `PythonTask` allows you to execute Python code in your task, while the `BashTask` provides an easy to use task for executing bash commands. In this tutorial we will use the `PythonTask` for all our tasks as it is the most flexible type of task. You can pretty much do whatever you like during the execution of a `PythonTask`.

Next, create the two tasks for our workflow. We are going to be boring here and call the first task `first_task` and the second task `second_task`:

```
first_task = PythonTask(name='first_task',
                        callback=print_first)

second_task = PythonTask(name='second_task',
                         callback=print_second)
```

The first argument `name` defines a name for the task so you can track the task more easily. We are using the name of the object here, but you can name the task whatever you think is appropriate. The second argument `callback` is a callable that is being run when the task is executed. This is the 'body' of the task and you are free to execute your own Python code here. In the spirit of boring names for our tutorial, we have named the callables: `print_first` and `print_second`. Of course, we haven't defined the callables yet, so let's do this next.

### Implement the callables

We will use functions as the callables for our `PythonTask` objects. The functions take a specific form and look like this:

```
def print_first(data, store, signal, context):
    print('This is the first task')
```

Add this code above your task instantiations. A callable for a `PythonTask` has four arguments. We will cover all four arguments in more detail in the following tutorial steps. So for now, you can safely ignore them. All we do in the body of the function is to print a simple string.

The callable for the second task is pretty much the same, we only change the name and the string that is printed:

```
def print_second(data, store, signal, context):
    print('This is the second task')
```

At this point we have the task objects that should be run and the code that should be executed for each task. We haven't defined the order in which we want the tasks to be run yet. This will happen in the next step.

### Arrange the tasks in a sequence

In Lightflow tasks are arranged in a Directed Acyclic Graph, or 'DAG' for short. While this might sound complicated, what it means is that all you do is to define the dependencies between the tasks, thereby building a network (also called graph) of tasks. The 'directed' captures the fact that the dependencies impose a direction on the graph. In our case, we want the `first_task` to be run before the `second_task`. Lightflow does not allow for loops in the task graph, represented by the word 'acyclic'. For example, you are not allowed to set up a graph in which you start with `first_task` then run `second_task` followed by running `first_task` again.

In Lightflow the `Dag` class takes care of running the tasks in the correct order. Import the `Dag` class at the top of your workflow file with:

```
from lightflow.models import Dag
```

Next, below your task object instantiations at the bottom of your workflow, create an object of the `Dag` class:

```
d = Dag('main_dag')
```

You have to provide a single argument, which is the name you would like to give to the Dag.

The `Dag` class provides the function `define()` for setting up the task graph. This is where the magic happens. Lightflow uses a Python dictionary in order to specify the arrangement of the tasks. The **key:value** relationship of a dictionary is mapped to a **parent:child** relationship for tasks, thereby defining the dependencies between tasks. For our simple, two task workflow the graph definition looks like this:

```
d.define({
    first_task: second_task
})
```

That's it! You have defined our first workflow and are now ready to run it.

### The complete workflow

Here is the complete workflow for this tutorial including a few comments:

```
from lightflow.models import Dag
from lightflow.tasks import PythonTask


# the callback functions for the task
def print_first(data, store, signal, context):
    print('This is the first task')

def print_second(data, store, signal, context):
    print('This is the second task')


# create the two task objects
first_task = PythonTask(name='first_task',
                        callback=print_first)

second_task = PythonTask(name='second_task',
                         callback=print_second)

# create the main DAG
d = Dag('main_dag')

# set up the graph of the DAG, in which the first_task has
# to be executed first, followed by the second_task.
d.define({
    first_task: second_task
})
```

#### Document the workflow

This step is optional, but highly recommended as it will help you remembering what the workflow does. We will add a title and a short description to the workflow. At the top of your workflow file add the following docstring:

```
""" Tutorial 1: a sequence of two tasks

This workflow uses two tasks in order to demonstrate
the basics of a workflow definition in Lightflow.
"""
```

Lightflow uses the first line of the docstring when listing all available workflows. Give it a go by changing to the directory where the configuration file is located and enter:

```
$ lightflow workflow list
tutorial01      Tutorial 1: a sequence of two tasks
```

Lightflow will list your workflow together with the short description you gave it.

#### Start a worker

Lightflow uses a worker based scheme. This means a workflow adds jobs onto a central queue from which a number of workers consume jobs and execute them. In order for Lightflow to run our workflow, it needs at least one running worker. Start a worker with:

```
$ lightflow worker start
```

This will start a worker, which then waits for the first job to be added to the queue. You can start as many workers as you like, but for now one worker is enough.

#### Run the workflow

With at least one worker running, we are ready to run our first workflow. You might need a second terminal in order to run the workflow as the first one is occupied running our worker. In your second terminal enter:

```
$ lightflow workflow start tutorial01
```

This will send our workflow to the queue. The worker will pick up the workflow and run it. The default logging level is very verbose so you will see the worker print out a lot of information as it executes the workflow.

You will see how the `first_task` is being executed first and prints the string "This is the first task", then followed by the `second_task` and the string "This is the second task".

Congratulations! You completed the first tutorial successfully.

### 1.3.3 Running in parallel

### 1.3.4 Making a decision

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search