kliko Documentation

Release 0.7.1

Gijs Molenaar

December 30, 2016

Contents

Contents:

Introduction

Kliko is a specification, validator and parser for the Scientific Compute Container specification. Kliko is written in Python.

This documentation is intended for the developer who wants to package up a piece of software into a Kliko container.

1.1 Installation

Development of Kliko is done on github. You can install Kliko inside a docker container or just on your system:

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

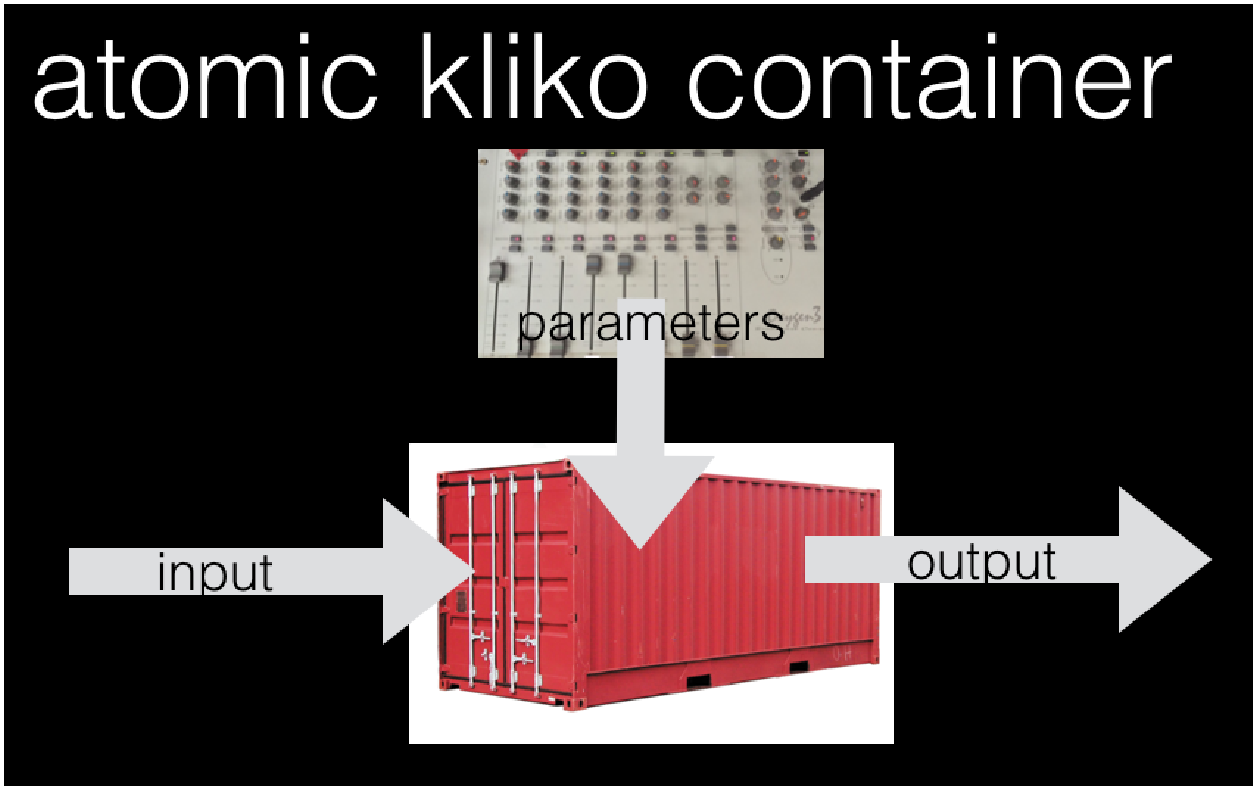
or from pypi:

```
$ pip install kliko
```

1.2 Why Kliko?

Kliko was born out of our needs to have a more formal and uniform way of scheduling batch compute tasks on arbitrary public and private cloud platforms. Docker is perfect for encapsulating and distributing software, but the input output flow is not defined. Kliko is an attempt to create a standard way to define compute input, output and parameters.

Kliko assumes your problem looks like this:



Kliko containers can be chained up in a sequence, for example using Luigi. An other use case is to simplify the parameterized scheduling of compute tasks using RODRIGUES.

1.3 Getting started

1.3.1 Creating a Kliko container

- Create a Docker container from your application
- Create a script `/kliko` in the container that can parse and use a `/parameters.json` file.
- Add a `kliko.yml` file to the root of the container which defines the valid fields in the parameters file.
- You can validate your kliko file with the `kliko-validate` script installed by the kliko Python library.

1.3.2 Running a kliko container

You can run a kliko container in various ways. The most simple way is to use the `kliko-run` script which is installed on your system when you install Kliko. Use `kliko-run <image-name> --help` to see a list of accepted parameters.

If you already have a parameters file you can also run the container manually:

```
$ docker run -v $(pwd)/parameters.json:/parameters.join:ro -v $(pwd)/input:/input:ro -v $(pwd)/output
```

Finally you can also run kliko images and visualise results using RODRIGUES, a web based kliko runner.

1.4 Contributing

Contributions are more than welcome! If you experience any problems let us know in the bug tracker. We accept patches in the form of github pull requests. Please make sure your code works with python 2 and python3, and is pep8 compatible. Also make sure the test suit actually passes all tests. We use docker in some of the tests so you need to have that installed and configured.

1.5 Testing

Note that before you run the test suite you have to create a `klikotest` docker image by running `make` in the `examples` folder.

CHAPTER 2

Terminology

2.1 Kliko

A specification which defines constrains on a docker container to aid in the scheduling of scientific compute tasks.

It is also a Python library that can be used to check if a container confirms the specification.

2.2 Kliko image

A Docker image confirming to the kliko specification. An image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. Images are read-only.

2.3 Kliko container

A container is an active (or inactive if exited) stateful instantiation of an image.

Read more about Docker terminology in the Docker glossary.

2.4 Kliko runner

Something that can run a kliko image. For example the `kliko-run` command line tool, or RODRIGUES.

2.5 The `/kliko.yml` file

A yaml formatted file confirming to the Kliko specification that defines the parameters a Kliko container is expecting. This is the file you want to create and add to your dockdr image if you want to create a Kliko container.

2.6 The `/parameters.json` file

A json encoded structure that contains all the parameter values for your compute task. This file is presented to your container at runtime by the container runner, for example RODRIGUES or Nextflow. The valid fields are defined by the Kliko image container and are defined in the `kliko.yml` file.

2.7 The `/param_files` folder

Files defined in the `kliko.yml` file and specified during runtime should be copied to the `param_file` folder by the kliko runner.

CHAPTER 3

The specification

- Kliko is based on standard docker containers
- A Kliko container should have a `/kliko.yml` fiel which defines the accepted parameters.
- A Kliko container should have a runable binary or script named `/kliko`. This will be the entrypoint for the Kliko runner.
- Logging should be written to STDOUT and STDERR.
- We define two types of compute containers, split IO and joined IO containers. For split IO Input files will be mounted read only into `/input`. Output file should be written to `/output`, which will be mounted by the host. For joined IO containers input & output is the /work folder which will be mounted RW.
- Parameters for the computation will be given when the container is run in the form of a file in json format called `/parameters.json`
- Fields with type file will enable supply of custom input files. these will be put in the `/input` folder.

3.1 The /kliko.yml file

The kliko file should be in YAML format and has these required fields:

3.1.1 schema_version

The version of the kliko specification. note that this is independent of the versioning of the Kliko library.

3.1.2 name

Name of the kliko image. For example `radioastro/simulator`. Optional.

3.1.3 description

A more detailed description of the image.

3.1.4 author

Who made the container. Optional.

3.1.5 email

email adres of the author. Optional.

3.1.6 url

Where to find the specific kliko project on the web.

3.1.7 io

Which IO mode to use, could be `join` or `split`. For split IO Input files will be mounted read only into `/input`. Output file should be written to `/output`, which will be mounted by the host. For joined IO containers input & output is the /work folder which will be mounted RW.

3.1.8 Sections

The parameters are grouped in sections. Sections are just lists of fields.

3.1.9 fields

A section consists of a list of fields.

3.1.10 field

each field has 2 obligatory keys, a `name` and a `type`. Name is a short reference to the field which needs to be unique. This will be the name for internal reference. The type defines the `type` of the field and can be one of `choice`, `string`, `float`, `file`, `bool` or `int`.

Optional keys are:

- **initial**: supply a initial (default) value for a field
- **max_length**: define a maximum length in case of string type
- **choices**: define a list of choices in case of a choice field. The choices should be a mapping
- **label**: The label used for representing the field to the end user. If no label is given the name of the field is used.
- **required**: Indicates if the field is required or optional
- **help_text**: An optional help text that is presented to the end user next to the field.

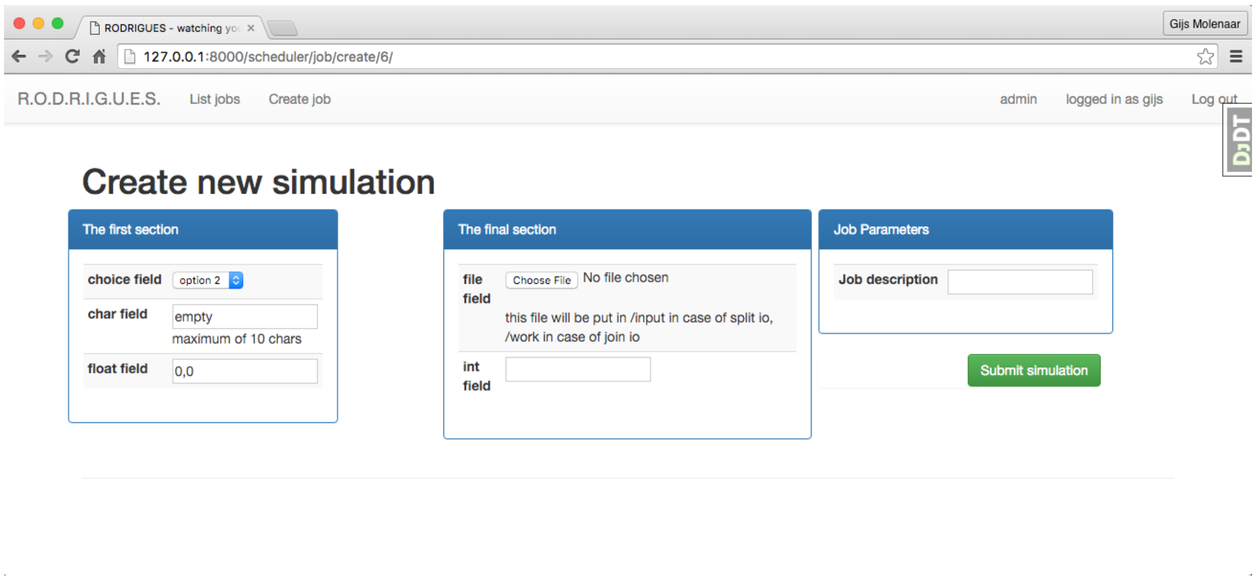
3.2 An example kliko.yml file

Below is an example kliko file.

```
schema_version: 3
name: kliko test image
description: for testing purposes only
url: https://github.com/gijzelaerr/kliko/tree/master/examples/fitsdoubler
io: split
```

```

sections:
  -
    name: section1
    description: The first section
    fields:
      -
        name: choice
        label: choice field
        type: choice
        initial: second
        required: True
        choices:
          first: option 1
          second: option 2
      -
        name: string
        label: char field
        help_text: maximum of 10 chars
        type: str
        max_length: 10
        initial: empty
        required: True
      -
        name: float
        label: float field
        type: float
        initial: 0.0
        required: False
  -
     name: section2
     description: The final section
     fields:
       -
         name: file
         label: file field
         help_text: this file will be put in /input in case of split io, /work in case of join io
         type: file
         required: True
       -
         name: int
         label: int field
         type: int
         required: True
```

Loading a Kliko container with the previous kliko file is loaded up in RODRIGUES will result in the form below:



Create new simulation

The first section

choice field	option 2
char field	empty maximum of 10 chars
float field	0,0

The final section

file field	Choose File No file chosen this file will be put in /input in case of split io, /work in case of join io
int field	

Job Parameters

Job description	

Processing this form will result in the following parameters.json file which is presented to the Kliko container on runtime:

```
{"int": 10, "file": "some-file", "string": "gijs", "float": 0.0, "choice": "first"}
```

CHAPTER 4

Inside the container

Inside the kliko container you can use the kliko library to validate the parameters file and read the settings.

4.1 validation and parsing

Validating and parsing the parameters is quite simple:

```
from kliko.validate import validate
parameters = validate()
```

This would open read and parse the files from the default locations. The parameters from `/parameters.json`, which are then validated against `/kliko.yml`.

4.2 parameter files

Files defined in the `kliko.yml` file and specified during runtime should be copied to the `param_file` folder by the kliko runner.

4.3 Environment variables

A kliko runner can have influence on the default location by setting environment variables. these variables are:

- `INPUT` (`kliko.input`) - controlling the input folder location, default `/input`
- `OUTPUT` (`kliko.output`) - controlling the input folder location, default `/output`
- `WORK` (`kliko.work`) - controlling the input folder location, default `/work`
- `PARAM_FILES` (`kliko.param_files`) - controlling the input folder location, default `/parame_files`
- `KLIKO_FILE` (`kliko.kliko_file`) - controlling the input folder location, default `/kliko.yml`
- `PARAM_FILE` (`kliko.param_file`) - controlling the input folder location, default `/parameters.json`

These

CHAPTER 5

Command Line Utilities

5.1 kliko-run

Use this to run the container. Use `kliko-run <image-name> --help` to see a list of accepted kliko paramaters, which are kliko container specific. You can override the default input, output (split io) and work (join io) folders using `--input`, `--output` and `--work` flags.

Attention: Always supply absolute paths to these flags, not absolulte. Docker doesn't work well with relative paths.

Note: On OSX Kliko-run will create a `parameters.json` file and a `param_files` folder in the current worker directory. Normally these are created in a temporary directory in your system, but since Docker on OSX doesn't mount the temporary folder into the docker virtual machine these files are inaccessable from within the docker engine and containers.

5.2 kliko-validate

Use this script to check if kliko container is valid.

CHAPTER 6

chaining containers

Kliko becomes more interesting in a multicontainer context. It is possible to chain the output of a container to the input of a next container. There are multiple ways to accomplish this.

6.1 The manual bash way

you can manually set the input and output folders of the kliko containers and call each consequtive step manually:

```
kliko-run kliko/simms --output simms  --tel meerkat
kliko-run kliko/meqtree-pipeliner --output meqtree-pipeliner --input simms
kliko-run kliko/wsclean --output wsclean --input meqtree-pipeliner
```

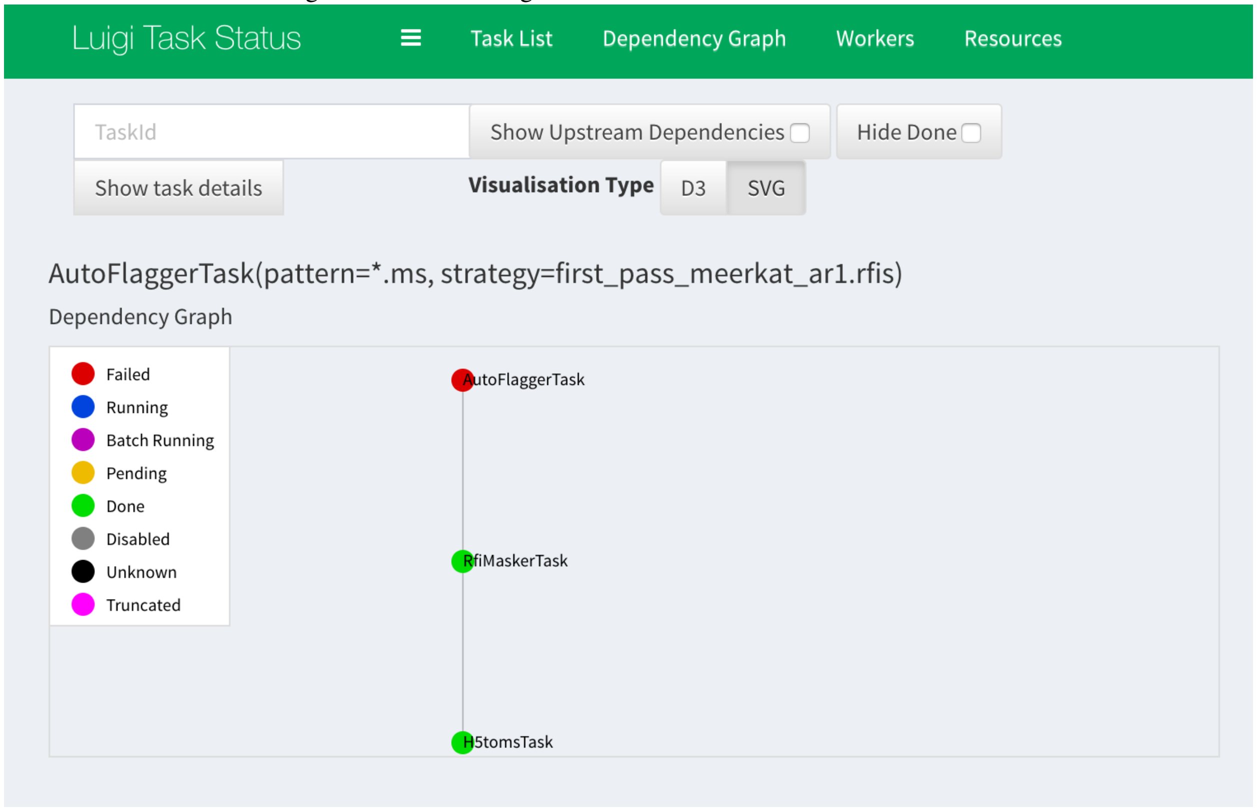
6.2 Using Luigi

Since Kliko 0.8 also has support for Luigi. Luigi is a Python package that helps you build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization, handling failures, command line integration, and much more.

Combinig Luigi and Kliko is quite simple, you need to define a `KlikoTask` and override the `image_name` method to define the Kliko Image name. You can then define the Task dependencies using the `requires()` method. Here is an example:

```python
from kliko.luigi_util import KlikoTask

class DownloadTask(KlikoTask):
    @classmethod
    def image_name(cls):
        return "vermeerkat/downobs:0.1"

class H5tomsTask(KlikoTask):
    @classmethod
    def image_name(cls):
        return "vermeerkat/h5toms:0.1"

    def requires(self):
        return DownloadTask(url='http://somewhere/somefile.h5', filename='1471892026.h5')

class RfiMaskerTask(KlikoTask):
    @classmethod
    def image_name(cls):
        return "vermeerkat/rfimasker:0.1"

    def requires(self):
        return H5tomsTask()

class AutoFlaggerTask(KlikoTask):
    @classmethod
    def image_name(cls):
        return "vermeerkat/autoflagger:0.1"

    def requires(self):
        return RfiMaskerTask(mask='rfi_mask.pickle')

class WscleanTask(KlikoTask):
    @classmethod
    def image_name(cls):
        return "vermeerkat/wsclean:0.1"

    def requires(self):
        return AutoFlaggerTask()
```

Which would look something like this in the Luigi web interface:



AutoFlaggerTask(pattern=*.ms, strategy=first_pass_meerkat_ar1.rfis)

Dependency Graph

6.3 Simple kliko chaining

If you don't want to use Luigi we also implemented simple container chaining with intermediate result caching in kliko. This will create a subfolder `.kliko` in your current working directory, containing subdirectories of the sha256 hash of the image. Each image hash folder will contain one or more subfolders which are named after the hash created from them specified parameters. If a Kliko chain is ran and the hash folders already exist the container is not ran but the results are passed to the next step in the chain.

Example:

```python
from kliko.chaining import run_chain
import docker

docker_client = docker.Client()

run_chain(
    (
        ('kliko/simms',  {'tel': 'meerkat'}),
        ('kliko/meqtree-pipeliner', {}),
        ('kliko/wsclean', {'weight': 'uniform'}),
    ),
    docker_client
)
```

CHAPTER 7

API

Kliko contains various helper functions to validate Kliko files, parameter files based on a kliko definition, generate command line interfaces and django forms from Kliko definitions.

7.1 Validation

Kliko and parameter validation related functions.

`kliko.validate.`**`convert_to_parameters_schema`**(*kliko*)

Convert a kliko schema into a validator for the parameters generated with a kliko schema.

Parameters **`kliko`** (`str`) – a kliko definition

Returns A structure for a pykwalify validator

`kliko.validate.`**`validate`**(*kliko_file=False*, *paramaters_file=False*)

Validate the kliko and paramaters file and parse the parameters file. Should be run inside the Kliko container.

Parameters

- **`kliko_file`** (`str`) – Path to a kliko file
- **`paramaters_file`** (`str`) – path to a parameters file

Returns The validated and parsed paramaters file

`kliko.validate.`**`validate_kliko`**(*kliko*, *version=3*)

validate a kliko yaml string

Parameters **`kliko`** – a parsed kliko object

Returns a (nested) kliko structure

Return type dict

Raises an exception if the string can't be parsed or is not in the following the Kliko schema

`kliko.validate.`**`validate_opened`**(*kliko*, *parameters*)

`kliko.validate.`**`validate_parameters`**(*parameters*, *kliko*)

validate a set of parameters given a kliko definition

Parameters

- **`parameters`** (`dict`) – A structure that should follow the given kliko structure
- **`kliko`** (`dict`) – A nested dict which defines the valid parameters in Kliko format

Returns the parsed parameters

Return type str

Raises an exception if the string can't be parsed or is not in the defining valid parameters

7.2 Command line interface generation

Command line utilities for Kliko

`kliko.cli.`**`command_line_run`**(*argv*)

`kliko.cli.`**`directory_exists`**(*path*)
check if a directory exists

`kliko.cli.`**`file_exists`**(*path*)
check if a file exists

`kliko.cli.`**`first_parser`**(*argv*)
This is only used when script in invoked with 0 or 1 args (should be kliko image name).

`kliko.cli.`**`generate_kliko_cli_parser`**(*kliko_data*, *parent_parser=None*)
Generate a command line parser from a Kliko structure.

Parameters **`kliko_data`** (`dict`) – A nested kliko structure

Returns a configured argument parser

Return type argparse.ArgumentParser

`kliko.cli.`**`second_parser`**(*argv*, *kliko_data*)
Used when kliko image is known, so we can extract the parameters.

7.3 Docker

Helper functions for using Kliko in combinaton with Docker

`kliko.docker_util.`**`extract_params`**(*docker_client*, *image_name*)

Parameters

- **`docker_client`** (`docker.docker.Client`) – a docker client object
- **`image_name`** (`str`) – name of the image to use for kliko.yml extraction

Returns content of the param schema

Return type str

7.4 Luigi

7.5 Chaining

`kliko.chaining.`**`run_chain`**(*steps*, *docker_client*, *kliko_dir=None*)
Run a chain of kliko containers. The output of each container will be attached to the input of the successive container.

Parameters

- **`steps`** (*`list`*) – a list of tuples, first element of tuple container name, second parameters dict
- **`docker_client`** (*`docker.Client`*) – a connection to the docker daemon
- **`kliko_dir`** (*`str`*) – a path to a workfolder for storing intermediate kliko results

7.6 Django

Helper functions for using Kliko in combinaton with Django

`kliko.django_form.`**`generate_form`**(*parsed*)

Generate a django form from a parsed kliko object

Parameters **`params`** – A parsed kliko file.

Returns form_utils.forms.BetterForm

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index