Porting Your Algorithm to Algorun Platform

Center of Quantitative Medicine – UConn Health

195 Farmington Avenue, Farmington - CT

Contents

Version Control

Date	Author	Contact	Changes
June 23, 2015	Ibrahim	abdelrahman.hosny@hotmail.com	- Initiated the document. - Added steps for algorun 0.9.1
June 30, 2015	Ibrahim	abdelrahman.hosny@hotmail.com	- Added introduction about Docker. - Added information about algorithm. - Enhanced the example.
July 8, 2015	Ibrahim	abdelrahman.hosny@hotmail.com	- Reflect changes in algorun_info

Introduction

Algorun is the base component of a data analytics platform that is built to support running and integrating algorithms written for algebraic modeling. It is built mainly to make ADAM[1] modules work together. In this document, we are illustrating how to make your algorithm run on Algorun in three easy steps. If you have more questions, go to http://algorun.org and ask for help.

Scope

The document covers the steps needed to run your algorithm on Algorun. By the end of this document, you should be able to run your algorithm as a web service with the least configuration possible. However, it does not describe how the platform is built nor the technical details of the internal workings. You may navigate to our website for more technical details.

Intended Audience

Information presented here are mainly intended for algorithms developers. Specially, developers who have an algorithm that fits in the domain of Algorun and wants to make it work on our platform.

What will the platform provide for you?

Using the platform, you are wrapping your algorithm dependencies along with all your algorithm code in a single container. Then, you can run your algorithm as a callable web service.

[1] ADAM: Analysis of Dynamic Algebraic Models. More info: https://github.com/PlantSimLab/ADAM

Before you start

Before you go with the rest of the document, we assume that you have a minimal knowledge with Docker[2] technology. If this is your first time to work with Docker, we are presenting a brief introduction about the technology and how to start with it.

What is Docker?

Docker is a platform that enables developers and system admins to ship their applications in the easiest way. In other way, Docker is a technology we use to wrap our applications in a container that we can move and run anywhere without having to make difficult configurations or dependencies installation. As long as you have Docker installed on a machine, you can run your application in a container; whatever your application does or dependencies it has. The following figure illustrates the main scheme.

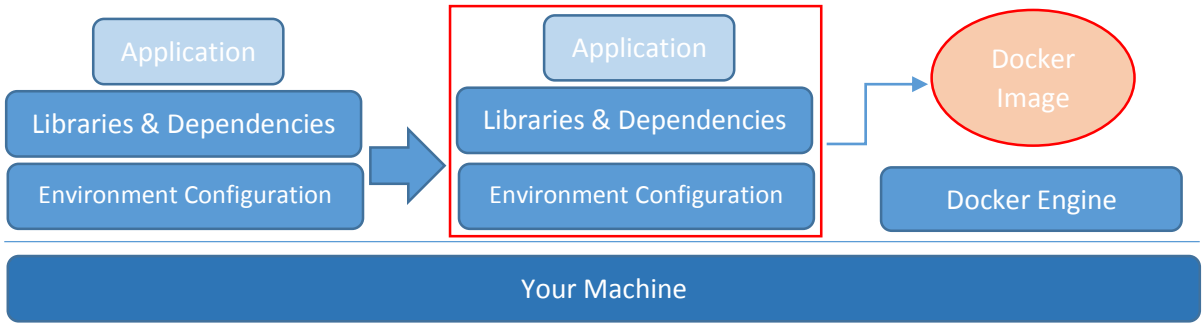

Figure 1 - Docker main scheme

Why do we use Docker?

So, in our work domain, we are using Docker for many reasons:

1- It enables us to wrap all application dependencies, libraries, source code and environment's default configuration in one image that can be run on any machine that has Docker engine installed. This is extremely useful in our problem domain as it comes over the problem of reproducibility of science. In other word, the problem of reproducing somebody else's results in computational sciences is still a barrier in improving research. Using Algorun, you can easily re-produce results very quickly.

2- It gives us the ability to transform an already-existing algorithm to work as a web interface without having to re-write algorithm code itself. This gives the opportunity for modelers to try an algorithm with no difficulty.

3- You do not have to write any additional code to port your algorithm to Algorun. Algorun Docker image will do the task for you.

Installing Docker

To install Docker on your machine, please navigate to http://docs.docker.com/mac/started/ and select your machine (Mac OS X, Linux or Windows) and follow the steps. You can make sure that the Docker is installed correctly by running:

```
docker run hello-world
```

[2] Docker is an open platform for distributed applications, developers and sys admins. More info: www.docker.com

Dockerfile

To build a Docker image, you can describe it in a Dockerfile. Find more information about Dockerfile on: https://docs.docker.com/reference/builder/

Algorun comes with a template Dockerfile that you will just edit; no need to write a one from scratch.

How to Port Your Algorithm to Algorun Platform

In this section, we will show you how to port your algorithm to Algorun. If you have not worked with Docker before, please refer to the previous section.

Step 1: Prepare your algorithm

- Your algorithm should accept the input as a file and produce the output to another file.
- Your algorithm should run as following:

```
command input_file [FLAGS] [-f output_file]
```

`command`	Mandatory	The command you use through terminal to run your algorithm
`input_file`	Mandatory	The input_file your algorithm read and process
`FLAGS`	Optional	If you have different modes for running the algorithm, you specify them as flags
`-f output_file`	Optional	If your command allow to specify a specific output file as a parameter, it should be in this format. If not, your algorithm should still write the output to a file (you will specify its name in the next step)

Step 2: Wrap your algorithm in a container

To make the process easy, we have provided you with a directory that contains all what you need. You will just edit the files on it. Go to our repository: https://github.com/algorun/algorun and clone it to your machine (You will just use 'try it' directory for now). When you open the directory, you should be able to figure out the folders as in the following figure:

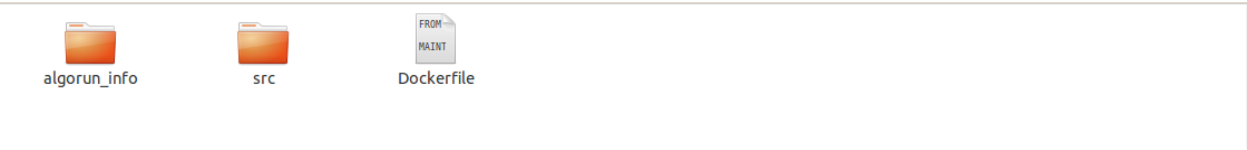

Figure 2 - Try Algorun directory

- First, you put all your algorithm source code in src folder.
- Second, you edit each file in the algorun_info folder. The information you put here will be displayed in the web page of your algorithm.
    - manifest.json: contains your algorithm name, a short description, a long description, authors, a web link and algorithm default parameters.
    - input_example.txt: give an example of how the input should be formatted.
    - output_example.txt: give an example of how the output looks like.

- The last step is to edit the Dockerfile. Open the file and follow the instructions in it. If you want to get a better understanding, you may navigate to the index in this documentation (You do not have to; comments in the Dockerfile are self-illustrative).

What are algorithm parameters?

In some cases, your algorithm might use some configuration. This configuration should be presented as parameters in a key-value pair **(parameter name, parameter value)**. To provide default configuration, we make use of environment variables to save the default values of these parameters. Your algorithm then reads default configuration from environment variables.

In the provided web interface, the user can easily change these parameters and run the algorithm again with the new configuration. You specify the parameters in the Dockerfile for your default configuration as well as in the manifest file for user editing. Note that if you do not include your parameters in the manifest file, your default configuration still apply, but cannot be changed by the user from the web interface.

Step 3: Build, run and test your container

- In you working directory, use the following command to build your container:
  `docker build -t <username>/<algorithm> .`
- Make sure that the build is successful.
- Use the following command to run your container:
  `docker run -d -p 31331:8765 <username>/<algorithm>`
- Open your web browser and go to http://localhost:31331 and see your algorithm in action.

> ***Tip:*** *you can also send an HTTP POST request to http://localhost:31331/do/run with parameter 'input' containing the test input to the algorithm. You should expect the output immediately as the output of the request.*

Example

In 'try it' directory we have added a very simple ruby application that reads an input, appends a string to it and send it back (not an algorithm; but simple for illustration). We have also edited all files in 'algo_run' info directory as a sample algorithm and then we added the instructions necessary in the Dockerfile. You can check the appendix for a detailed example. Lastly, we build and run our container as indicated in the previous section.

Appendix

Algorun Dockerfile Description

Algorun container has `/home/algorithm/` as its working directory. It comes pre-built with the required web API that wrap your algorithm. Let's have a look at the Dockerfile line by line to understand what is going on.

```
FROM ahosny/algorun:0.9.1
```

This is the base image that you inherit the web API from. It is a ubuntu:14.04 image that uses node.js to serve the web API and the user interface.

```
# Don't forget to fill files in algorun_info folder and put your source code in src folder
ADD ./algorun_info /home/algorithm/web/algorun_info/
ADD ./src /home/algorithm/src/
```

The next step is to add your algorithm code and information to the container. Leave these two lines unchanged.

```
# Add any algorithm dependencies here
RUN apt-get -y curl
```

Your algorithm may require some dependencies and installations to run. Make sure you install them at this step

---

```
# Add your command to execute an algorithm as environment variable EXEC_COMMAND
ENV EXEC_COMMAND "ruby test.rb"
```

Remember what you did in step 1? Yes, you prepared it the command to be like that:
`command` `input_file [FLAGS] [-f output_file]`
In this line, you specify the command part as environment variable. **Only** the command. The input_file is automatically generated through Algorun from the HTTP POST request parameter 'input'

```
# Add your command options (if any) as environment variable COMMAND_OPTIONS
ENV COMMAND_OPTIONS "-c"
```

Again, in the command you specified in part 1, this line represents **`[FLAGS]`** options.

```
# If your code outputs results to a hard-coded file name, specify it as environment variable
OUTPUT_FILE_NAME
ENV OUTPUT_FILE_NAME "output.txt"
```

If your command in step 1 is flexible enough to specify –f flag, you may ignore this step. If not and your algorithm outputs your result to a hard-coded file, you should provide the file name to be able to locate it after the algorithm runs.

```
# If your algorithm uses some default configuration, pass them as environment variables
ENV PARAM_* value
```

If you want to pass default configuration to your algorithm, such as parameters, use the environment variable PARAM_*
The * indicated parameter name. Note that it is totally your responsibility to read these environment variables in your code. We do not use it.
