Screwjack Documentation

Release 0.1.1

Xiaolin Zhang

June 19, 2014

Contents

ScrewJack is a tiny command line tool for manipulating modules.

CHAPTER 1

Getting Started

If you never used `screwjack` before, you should read the *Getting Started with Screwjack(Basic)* guide to get familiar with `screwjack` and its usage.

1.1 Introduction

ScrewJack is a tiny command line tool for manipulating modules.

1.1.1 Basic Concepts

Screwjack is a utility for helping module designers compose modules. Modules are defined by a file named `spec.json`. Here is the a example of `spec.json`: n

```
{
    "Name": "SVM",
    "Description": "A simple SVM",
    "Version": "0.1",
    "Cmd": "/usr/bin/python main.py",
    "Param": {
        "C": {
            "Default": "",
            "Type": "string"
        }
    },
    "Input": {
        "X": ["csv"],
        "Y": ["csv"]
    },
    "Output": {
        "MODEL": ["model.svm"]
    }
}
```

In short, screwjack is a utility work around `spec.json`. Typically, there are 5 steps to write a module. The following tutorial will show details steps.

1. Initialize a module
2. Add Inputs/Outputs/Params
3. Fill your code implementation

4. Test module
   (a) Test in **local**
   (b) Test in **docker**
5. Submit module

1.1.2 Installation

Screwjack will depends on docker, so you should install docker first.

Install docker

A module developing environment need docker. Follow the link to install docker for your linux distribution: http://docs.docker.io/installation/.

After that, don't forget add yourself into 'docker' group. For example, in Ubuntu, you can do it like this:

```
sudo usermod -aG docker your_linux_username
```

Install screwjack

You can get screwjack directly from PyPI:

```
pip install -U screwjack
```

Setup screwjack

Before you using screwjack, you should set your username. You can either set environment variable:

```
export DATACANVAS_USERNAME=your_username
```

Or, you can put your username into `$HOME/.screwjack.cfg`:

```
[user]
username = your_username
```

Or, you can add `--username` option for screwjack like following:

```
screwjack --username=your_username init
screwjack --username=your_username param_add
screwjack --username=your_username input_add
screwjack --username=your_username output_add
```

1.2 Getting Started with Screwjack(Basic)

Before you trying following, you should ensure screwjack is installed. Please refer *Introduction* for detail installation steps.

1.2.1 Step 1: Initialize a module

First, assume you want create a **basic** module, which is a template with basic functinality. If you interested in writing a 'Hive' module, please refer to *Getting Started with Screwjack(Hive)*.

So, you can create a basic module with `screwjack`:

```
screwjack init basic --name="SVM" --description="A simple SVM"
```

Then, it will prompt to setup other options, like the following. In this tutorial, we will use scikit-learn, which are packed in base image `zetdata/sci-python:2.7`.

```
Module Version [0.1]:
Module Entry Command [/usr/bin/python main.py]:
Base Image [zetdata/ubuntu:trusty]: zetdata/sci-python:2.7
Sucessfully created 'svm'
```

Or, you can use single command to do this:

```
screwjack init basic --name=SVM --description="A simple SVM" --version="0.1" --cmd="/usr/bin/python m
```

Now, you will get a directory with initial verison of basic module:

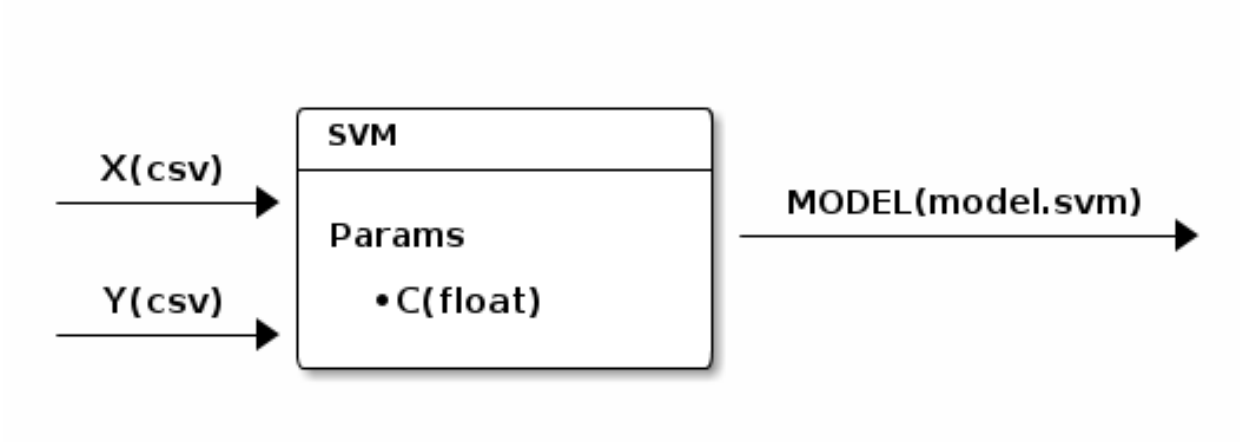
```
svm
|-- Dockerfile
|-- main.py
|-- spec.json
`-- specparser.py

0 directories, 4 files
```

Then you should change to the directory of the new module, the following steps will assume we are working at that directory.

```
cd svm
```

1.2.2 Step 2: Add Input/Output/Param

Image we want create a module with two *Input*s, one **Output**, and one parameter. Just like the following diagram shows:



Now we can add a parameter using the following command:

```
screwjack param_add C
```

And, we add two Inputs by the following commands. The first argument **X** means the name of the input/output, and the second argument **csv** means the type for this input/output. A type can be any string, like "csv", "hive.hdfs.table:sub:*x*". For more information about types, please follow *Input/Output Types*.

```
screwjack input_add X csv
screwjack input_add Y csv
```

Finally, a Output:

```
screwjack output_add model model.svm
```

1.2.3 Step 3: Fill your code implementation

Now, you can write your awesome implementation now:

```
vim main.py
```

In this tutorial, we would like implement our `main.py` like this:

```
from specparser import get_settings_from_file

from sklearn.svm import LinearSVC
import numpy as np
import pickle

def main():
    settings = get_settings_from_file("spec.json")
    X = np.genfromtxt(settings.Input.X, delimiter=',', skip_header=1)
    Y = np.genfromtxt(settings.Input.Y, delimiter=',', skip_header=1)
    svc = LinearSVC(C=float(settings.Param.C))
    svc.fit(X,Y)
    with open(settings.Output.MODEL, "w") as f:
        pickle.dump(svc, f)
    print("Done")

if __name__ == "__main__":
    main()
```

If you want add additional files for this module, don't forget add files in `Dockerfile`.

```
vim Dockerfile
```

For example, if you have additional file, you should append the following line into `Dockerfile`:

```
ADD your_additional_file /home/run/
```

In the case if you want add additional folder, you should append a line like this:

```
ADD your_additional_folder /home/run/your_additional_folder
```

For more information about `Dockerfile`, please reference Dockerfile.

1.2.4 Step 4.1: Test in local

After write own implementation into this module, we might want test it. The `screwjack run` subcommands are design for this.

```
screwjack run local --help

Usage: screwjack run local [OPTIONS]
Options:
  --param-C TEXT  Param(string)
  --X TEXT        Input
  --Y TEXT        Input
  --MODEL TEXT    Output
  --help          Show this message and exit.
```

Now, we can test our module in local environment, which is very close to your developing environment.

```
screwjack run local --param-C=0.1 --X=a.csv --Y=b.csv --MODEL=tmp.model
```

1.2.5 Step 4.2: Test in docker

Then, we can try to execute module by docker:

```
screwjack run docker --param-C=0.1 --X=a.csv --Y=b.csv --MODEL=tmp.model
```

1.2.6 Step 5: Submit module

You should provide the URL of `spec_server` to submit:

```
screwjack submit
```

1.3 Getting Started with Screwjack(Hive)

This is a show case for the usage of screwjack hive runtime. This section we will create a EMR hive module to get the topN hot tokens in search queries.

At first, thanks AOL share their query log online [1]. We download a piece of them from http://www.infochimps.com/datasets/aol-search-data as the test data in this demo. The data looks like below:

```
AnonID  Query   QueryTime   ItemRank    ClickURL
142 rentdirect.com  2006-03-01 07:17:12
142 www.prescriptionfortime.com 2006-03-12 12:31:06
142 staple.com  2006-03-17 21:19:29
142 staple.com  2006-03-17 21:19:45
142 www.newyorklawyersite.com   2006-03-18 08:02:58
142 www.newyorklawyersite.com   2006-03-18 08:03:09
142 westchester.gov 2006-03-20 03:55:57 1   http://www.westchestergov.com
142 space.comhttp   2006-03-24 20:51:24
142 dfdf    2006-03-24 22:23:07
142 dfdf    2006-03-24 22:23:14
142 vaniqa.comh 2006-03-25 23:27:12
142 www.collegeucla.edu 2006-04-03 21:12:14
142 www.elaorg  2006-04-03 21:25:20
142 207 ad2d 530    2006-04-08 01:31:04
142 207 ad2d 530    2006-04-08 01:31:14 1   http://www.courts.state.ny.us
142 broadway.vera.org   2006-04-08 08:38:23
```

[1] G. Pass, A. Chowdhury, C. Torgeson, "A Picture of Search" The First International Conference on Scalable Information Systems, Hong Kong, June, 2006.

```
142 broadway.vera.org   2006-04-08 08:38:31
142 vera.org    2006-04-08 08:38:42 1   http://www.vera.org
142 broadway.vera.org   2006-04-08 08:39:30
142 frankmellace.com    2006-04-09 02:19:24
142 ucs.ljx.com 2006-04-09 02:20:44
142 attornyleslie.com   2006-04-13 00:25:27
142 merit release appearance    2006-04-22 23:51:18
.........................
```

1.3.1 Step 0: Initialize a hive module

```
screwjack init emr_hive -n hot_token_topN_on_emr -d "Get hotest token in search engine query log."
```

When prompt for `Module Version` and other options, you can press ENTER to use default options.

```
Module Version [0.1]:
Module Entry Command [/usr/bin/python main.py]:
Base Image [zetdata/ubuntu:trusty]:
init emr_hive
Sucessfully created 'hot_token_topn_on_emr'
```

After that, you will get a directory which its name is `hot_token_topn_on_emr`.

1.3.2 Step 1: Add the input/output and parameter to this module.

```
screwjack input_add query_log_s3_dir hive.s3.id_query_querytime
screwjack output_add hot_token_topN_s3_dir hive.s3.table.token_count
screwjack param_add topN string
```

Here `query_log_dir` is the hdfs dirctory which contain the raw data. The schema of the data is id,query and querytime. The `hot_token_topN` is the hive table name we gonna dump our result in.

1.3.3 Step 2: (optional) Make the UDF help explore the query into tokens

This step is optional. When you want a UDF in your module. Here is a example at: example-modules. After you build the jar of your UDF, you should put it into `./resource/udfs`. Then, the HiveRuntime can automatically upload files s3.

1.3.4 Step 3: Write the hive script.

Then, we can open `main.hql` to fill our code like this:

```
set hive.base.inputformat=org.apache.hadoop.hive.ql.io.HiveInputFormat;

CREATE TEMPORARY FUNCTION splitword AS 'com.your_company.hive.udtf.SplitWord';

--CREATE OUTPUT TABLE

DROP TABLE IF EXISTS hot_token_topN_table;
CREATE EXTERNAL  TABLE hot_token_topN_table
(
        token STRING,
        freq  INT
```

```
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION '${OUTPUT_hot_token_topN_s3_dir}';

--CREATE AN EXTERNAL TABLE TO LOAD THE QUERY DATA
DROP TABLE IF EXISTS query;
CREATE EXTERNAL TABLE query
(
        id STRING,
        site STRING,
        timestp TIMESTAMP
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION '${INPUT_query_log_dir_s3_dir}';

INSERT OVERWRITE TABLE hot_token_topN_table
SELECT token, freq FROM
(
        SELECT token,count(1) AS freq FROM
        (
                SELECT splitword(site) AS token FROM query
        )token_table
        GROUP BY token
)token_frep
ORDER BY freq DESC LIMIT ${PARAM_topN};
```

You could reference the input parameter defined with screwjack by `${INPUT_inputname}`, such as `${INPUT_query_log_s3_dir}` in this case. Output parameter by `${OUTPUT_outputname}`, such as `${OUTPUT_hot_token_topN_s3_dir}` in this case. Parameter by `${PARAM_paramname}`, such as `${PARAM_topN}` in this case.

1.3.5 Step 4: Test locally

Before test, we need to upload the sample data to S3. Here we put them on `s3://get-hot-token-kk/input/query`. As module take input from its precursor, when we do test, we need to feed it by ourself. We touch the input parameter file and output parameter file to contain the input parameter and the output result. In this case, our input parameter is the s3 directory which contains the query logs. So create the file `./input.param` and write `s3://get-hot-token-kk/input/query` into. Then create an output param to recive the output.

```
screwjack run local
```

Then type the corresponding parameter to run the test.

```
Param 'FILE_DIR' [./resources/files]:
Param 'UDF_DIR' [./resources/udfs]:
Param 'AWS_ACCESS_KEY_ID' []: YOUR_AWS_ACCESS_KEY
Param 'AWS_ACCESS_KEY_SECRET' []: YOUR_AWS_ACCESS_KEY_SECRET
Param 'S3_BUCKET' []: get-hot-token-kk
Param 'AWS_Region' []: us-east-1
Param 'EMR_jobFlowId' []: YOUR_EMR_JOB_FLOW_ID
Param 'topN' []: 10
Input 'query_log_s3_dir': input.param
Output 'hot_token_topN_s3_dir': output.param
```

During the test, if any error or bug emerge, you could modfiy your udtf and hive script according to the propted log. If everything is ok, the defined output parameter will be created and written into the `output.param`. If the test finished successfully, we could get the a s3 directory in output.param. Here we get `s3://get-hot-token-kk/zetjob/your_username/job456/blk789/OUTPUT_hot_token_topN_s3_dir`. Let us open the file on s3 and we get the hotest 10 token among the query.

```
of,110575
-,104052
in,91521
the,82961
for,70107
and,66675
to,45168
free,45149
a,36220
google,34970
```

1.3.6 Step 5: Test in docker

```
screwjack run docker
```

This step is to test whether the module could correctly run in a docker image. At first, screwjack will help to build a specific image with a hive runtime in. Then it will test your script and udfs in this image. If it turns out to be a success, we get this module ready to run online.

1.4 Type of initial modules

Writing module seems hard, so we trying to design a set of templates according to different screnario:

Name	Command
Basic	`screwjack init basic`
Hive(CDH4)	`screwjack init hive`
PIG(CDH4)	`screwjack init pig`
EMR(Hive)	`screwjack init emr_hive`
EMR(Pig)	`screwjack init emr_pig`

1.5 Base Images

1.5.1 Why base images?

Docker offers revolutiony platform to build, ship containerized images. The official images may not match our requirements for packaging modules. So, our base images are design according to following reasons:

Base images provide minimum requirements for specific usage. For example, `zetdata/cdh:4` are base images for CDH4 hadoop cluster. And `zetdata/sci-python:2.7` can provide scikit-learn toolkit for python data scientists.

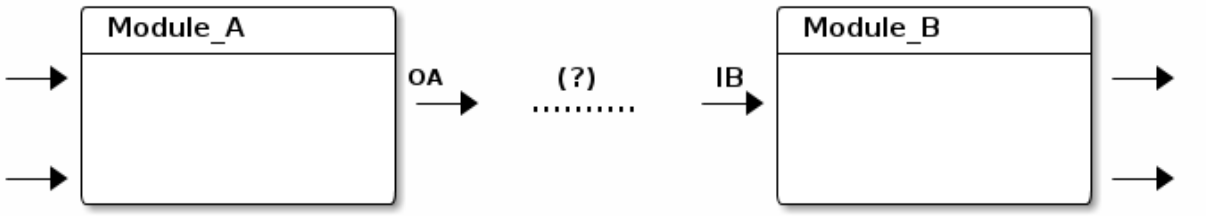
Can also save bandwith. Of course, you can build your own module from scratch. Docker offers incremental pulling, which will pull differental images from remote registry server. So, it may save bandwith and time if we share same base images.

All scripts to build base images are released at github repo, and images are pushed to official docker registry.

1.5.2 Hierarchy of base images

1.6 Input/Output Types

1.6.1 Why types?



When we connect two pins between modules, the question arise that, can we connect them? As example in previous diagram shows, if `Module_A`'s output **OA** will write csv, and `Module_B`'s input want read tsv, our task will failed because of mismatch of types.

So, we borrow a simple type system like Union Type. It will help module designers to constraint module's inputs and outputs. In datacanvas frontend, we will also do type checking to help user's discover mismatch between input and output during design stage the work flow.

Informally, a simple-type is a just a string, for example, "csv", "csv.salary.table", "hive.table.tf". And, simple-type is case sensitive. Also, Input/Output type is a set of simple-type. So, two Input/Output pins can connect, if and only if the intersect of two types are not empty.

Type system will be helpful if it is applied properly. Some wrong way to use type system are:

1. Use same type for a lot of modules. This will "turn off" type checking.
2. Choose type name wisely. Some type name like, "hive.table.A" are less informative. It may be less helpful for using that module.

1.6.2 Type List

Hive

1.7 Runtime System

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*