Preparing Tools

The Brown Dog Data Transformation Service (DTS) is a highly extensible/distributed service providing a uniform means of managing and accessing transformation capabilities within the web. Utilized tools can come in the form of command line applications, GUI driven applications, libraries, and/or other services. Here we go over the process of preparing a new transformation tool, either an extractor or a converter, for usage with the DTS.

Using the BD Development Base

BD-base runs the necessary dockerized Brown Dog Data Transformation Service components (Clowder, Polyglot, Fence, RabbitMQ, MongoDB, Redis, an example extractor, an example converter, and the BD CLI) allowing a developer to get up and running more quickly as they create and debug new extractors/converters. You can get the BD-base by cloning the git repo:

```
git clone https://opensource.ncsa.illinois.edu/bitbucket/scm/bd/bd-base.git
```

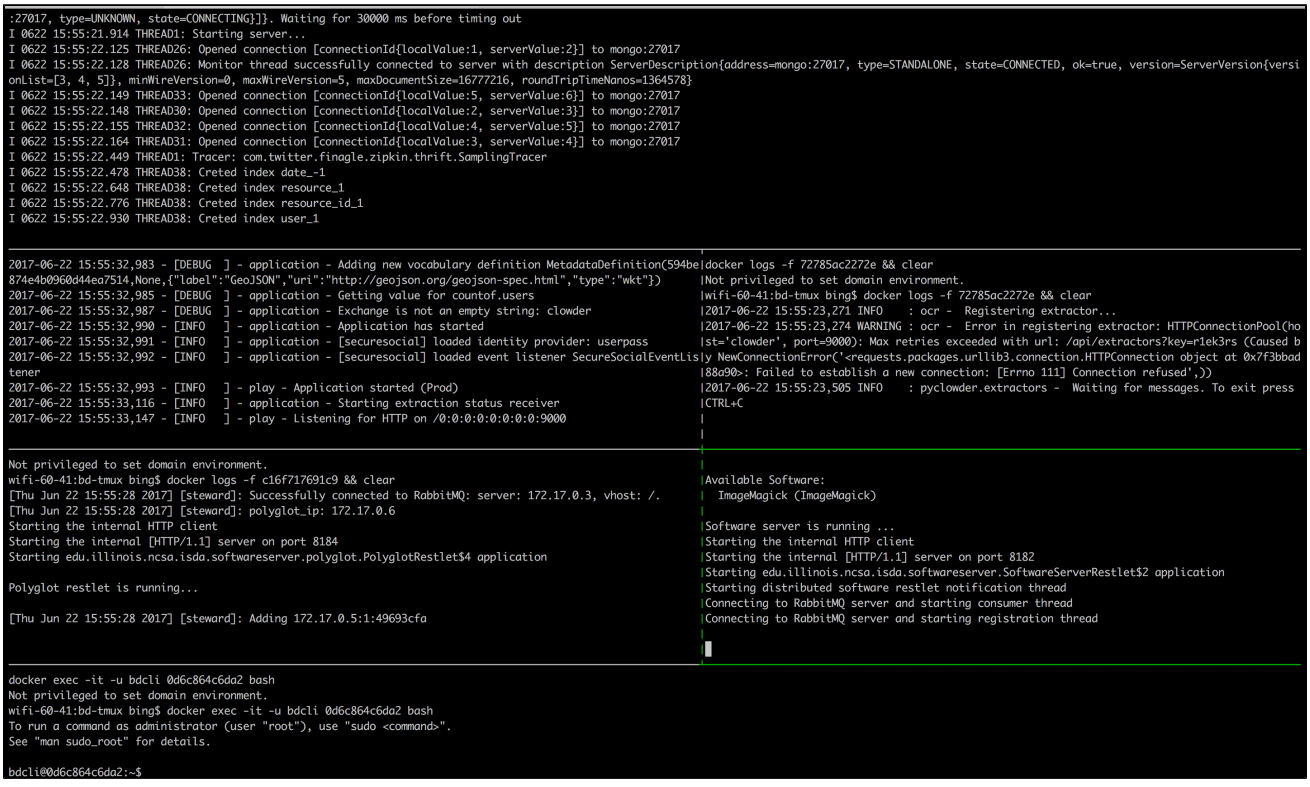
or download the VirtualBox VM image and run it:

https://browndog.ncsa.illinois.edu/downloads/bd-base.ova

After downloading BD-base, users can simply run the bash script in the command-line to start up the BD development base.

```
cd bd-base
./bd
```

The BD-base script will split your terminal into panes and start each of the services needed for the Brown Dog DTS. This provides a useful and convenient way to view the logs of running services in panes.



Users can switch between panes using Tmux commands. The panes are as follows: Fence (top), Clowder (middle-left), example extractor (middle-right), Polyglot (middle-left), example convert (middle-right), and the BD CLI (bottom). In the bottom pane, users can run BD-CLI commands to interact with the Brown Dog Data Transformation Service (username: bd, password: browndog):

```
bdcli@2f09151958f8:~$ bd -o bmp -b http://fence:8080 sample.jpg
Brown Dog API URL: http://fence:8080
Key not available. Getting new key.
Username: fence
Password:
Key: 6e8e0e0d-09de-49a9-9a2a-96b50a166d1a
Token expired or not available. Getting new token.
Token: 1219d9b4-311a-453c-86c5-d61a2125d1b1
sample.bmp
bdcli@2f09151958f8:~$ ls
sample.bmp  sample.jpg  supervisor
```

```
bdcli@f95d84b0ecec:~$ bd -v -b http://fence:8080 sample.jpg
Brown Dog API URL: http://fence:8080
Key not available. Getting new key.
Username: fence
Password:
Key: be3c6ffd-6151-430d-8683-2e966c412c32
Token expired or not available. Getting new token.
Token: 234d2d20-abde-4e7b-83ba-ad0ae5b2a955

{"metadata.jsonld": [{"content": {"ocr_text": "EB BROWSER MOSAIC THE FIRST POPULAR FOR THE WORLD WIDE BY MARC AND
REESSEN BINA THE NATIONAL CENTER COMPUTING APPLICATIONS NCSM 1993 RELEASE TO THE PUBLIC INTERNET USERS EASY ACCES
S TO SOURCES OF INFORMATION Wyn HAVE TRANSFORMED THE INFORMATION UNIVERSITY OF"}, "@context": ["https://clowder.n
csa.illinois.edu/contexts/metadata.jsonld", {"ocr_text": "http://clowder.ncsa.illinois.edu/ncsa.image.ocr#ocr_tex
t"}], "created_at": "Tue Jun 27 15:02:40 UTC 2017", "agent": {"extractor_id": "https://clowder.ncsa.illinois.edu/
clowder/api/extractors/ncsa.image.ocr", "@type": "cat:extractor", "name": "https://clowder.ncsa.illinois.edu/clow
der/api/extractors/ncsa.image.ocr"}}], "versusmetadata": [], "tags": [], "filename": "sample.jpg", "technicalmeta
data": [], "id": "59527389e4b005e57b632f98"}

sample.json     [OK]
Elapsed time: 8 seconds
```

CTRL-b <arrow key> will navigate panes.
Exit the bd-base session by typing CTRL-b then :kill-session.
NOTE: There is a .tmux.conf file included in bd-base. If you copy this file into your home directory before starting a bd-base session you will be able to navigate panes via the mouse and end the session by typing CTRL-b then CTRL-c.

Extractors

Here we describe the process for taking a working piece of code and deploying it as a Brown Dog extractor. For simplicity, it is assumed that the method can be invoked from a single call. In this example, we are using the python extractor wrapper and will invoke a python function. In a very similar fashion, a method developed in a language other than python can be invoked using subprocess.

The main steps:

1. Wrap the tool for use as an extractor in Clowder (and through that Brown Dog)
2. Dockerize the extractor
3. Deploy the extractor
4. Add the extractor to the Tools Catalog

A few assumptions are that you have a tool that extracts some kind of metadata from a file or dataset and that you have installed Python, Git, and Docker as well other specific software needed by your extractor (if any) on your computer.

1. Install pyClowder2

Install pyClowder2, which is a Python library that helps to easily communicate with Clowder - the backend service of Brown Dog which handles extractions. The advantage of using this library is that it manages all communications with Clowder and RabbitMQ (the distributed messaging bus) and the developer doesn't have to take care of such tasks. Needless to say, an extractor can also be written in native Python without the use of pyClowder2, but it could be more

```
pip install --upgrade pip
pip install -r https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/raw/requirements.
txt git+https://opensource.ncsa.illinois.edu/bitbucket/scm/cats/pyclowder2.git
```

2. Get Your Code Together

We have developed a template extractor written in Python. It is a simple word count extractor that counts lines, words, and characters in a text file. Clone the template extractor and rename the directory to an appropriate name that reflects the purpose of your extractor.

```
git clone https://opensource.ncsa.illinois.edu/bitbucket/scm/bd/extractors-template.git
mv extractors-template/ <your_extractor_name>
cd <your_extractor_name>
```

Make changes to extractors.py (main program). Consider the process_file method as the main method of an extractor and accordingly it needs to contain the main logic. You can call other methods in your python code from this method after importing necessary modules into this file.

```
# ----------------------------------------------------------------------
# Process the file and upload the results
def process_file(parameters):
    global extractorName

    inputfile = parameters['inputfile']

    # Call word count command
    result = subprocess.check_output(['wc', inputfile], stderr=subprocess.STDOUT)
    (lines, words, characters, filename) = result.split()

    # Context url
    context_url = 'https://clowder.ncsa.illinois.edu/contexts/metadata.jsonld'

    # Store results as metadata
    metadata = \
        {
            '@context': [
                context_url, {
                    'lines': 'http://clowder.ncsa.illinois.edu/' + extractorName + '#lines',
                    'words': 'http://clowder.ncsa.illinois.edu/' + extractorName + '#words',
                    'characters': 'http://clowder.ncsa.illinois.edu/' + extractorName + '#characters'
                }
            ],
            'attachedTo': {
                'resourceType': 'file', 'id': parameters["fileid"]
            },
            'agent': {
                '@type': 'cat:extractor',
                'extractor_id': 'https://clowder.ncsa.illinois.edu/clowder/api/extractors/' + extractorName
            },
            'content': {
                'lines': lines,
                'words': words,
                'characters': characters
            }
        }
    print metadata

    # Upload metadata
    extractors.upload_file_metadata_jsonld(mdata=metadata, parameters=parameters)
```

3. Edit extractor_info.json

This file contains information about the extractor in JSON-LD format. Update all relevant fields as needed.

```
{
    "@context": "http://clowder.ncsa.illinois.edu/contexts/extractors.jsonld",
    "name": "wordCount",
    "version": "1.0",
    "description": "Brown Dog template extractor",
    "author": "Rob Kooper <kooper@illinois.edu>",
    "contributors": ["Gregory Jansen <jansen@umd.edu>", "Sandeep Puthanveetil Satheesan <sandeeps@illinois.edu>"],
    "contexts": [],
    "repository": {"repType": "git", "repUrl": "https://opensource.ncsa.illinois.edu/bitbucket/scm/bd/extractors-template.git"},
    "external_services": [],
    "dependencies": [],
    "bibtex": []
}
```

4. Configuration Parameters

Extractors obtain the configuration details required to connect to RabbitMQ, Clowder, etc., either from command-line arguments or environment variables. If you look at the Dockerfile inside the termplate extractor directory you can see some of the environment variables being set. For the purpose of running your extractor using BD Development Base, you DO NOT have to change anything.

Note: The remaining part of this section is relevant ONLY if you want to run your extractor against another production instance at some other location. Otherwise, you can skip and continue reading the next section.

If you are planning to run your extractor using Docker, you will need to modify the Dockerfile to set the environment variables as required.

```
# Setup environment variables. These are passed into the container. You can change
# these to your setup. If RABBITMQ_URI is not set, it will try and use the rabbitmq
# server that is linked into the container. MAIN_SCRIPT is set to the script to be
# executed by entrypoint.sh. REGISTRATION_ENDPOINTS should point to a central clowder
# instance, for example it could be https://clowder.ncsa.illinois.edu/clowder/api/extractors?key=secretKey

ENV RABBITMQ_URI="" \
    RABBITMQ_EXCHANGE="clowder" \
    RABBITMQ_QUEUE="ncsa.wordcount" \
    REGISTRATION_ENDPOINTS="https://clowder.ncsa.illinois.edu/extractors" \
    MAIN_SCRIPT="wordcount.py"
```

Otherwise, if you run your extractor as a standalone program (outside of Docker), you will need to set the relevant command-line arguments. You can get a list of these parameters by running your extractor with the help option (-h, --help).

```
bd@bd-base:~/extractors-template$ ./wordcount.py -h
usage: wordcount.py [-h] [--connector [{RabbitMQ,HPC,Local}]]
                    [--logging [LOGGING]] [--num [NUM]]
                    [--pickle [HPC_PICKLEFILE [HPC_PICKLEFILE ...]]]
                    [--register [REGSTRATION_ENDPOINTS]]
                    [--rabbitmqURI [RABBITMQ_URI]]
                    [--rabbitmqQUEUE [RABBITMQ_QUEUENAME]]
                    [--rabbitmqExchange [RABBITMQ_EXCHANGE]]
                    [--mounts MOUNTED_PATHS]
                    [--input-file-path INPUT_FILE_PATH]
                    [--output-file-path OUTPUT_FILE_PATH] [--sslignore]
                    [--version] [--no-bind]

WordCount extractor. Counts the number of characters, words and lines in the
text file that was uploaded.

optional arguments:
  -h, --help            show this help message and exit
  --connector [{RabbitMQ,HPC,Local}], -c [{RabbitMQ,HPC,Local}]
                        connector to use (default=RabbitMQ)
  --logging [LOGGING], -l [LOGGING]
                        file or url or logging coonfiguration (default=None)
  --num [NUM], -n [NUM]
                        number of parallel instances (default=1)
  --pickle [HPC_PICKLEFILE [HPC_PICKLEFILE ...]]
                        pickle file that needs to be processed (only needed
                        for HPC)
  --register [REGSTRATION_ENDPOINTS], -r [REGSTRATION_ENDPOINTS]
                        Clowder registration URL (default=)
  --rabbitmqURI [RABBITMQ_URI]
                        rabbitMQ URI
                        (default=amqp://guest:guest@127.0.0.1/%2f)
  --rabbitmqQUEUE [RABBITMQ_QUEUENAME]
                        rabbitMQ queue name (default=ncsa.wordcount)
  --rabbitmqExchange [RABBITMQ_EXCHANGE]
                        rabbitMQ exchange (default=clowder)
  --mounts MOUNTED_PATHS, -m MOUNTED_PATHS
                        dictionary of {'remote path':'local path'} mount
                        mappings
  --input-file-path INPUT_FILE_PATH, -ifp INPUT_FILE_PATH
                        Full path to local input file to be processed (used by
                        Big Data feature)
  --output-file-path OUTPUT_FILE_PATH, -ofp OUTPUT_FILE_PATH
                        Full path to local output JSON file to store metadata
                        (used by Big Data feature)
  --sslignore, -s       should SSL certificates be ignores
  --version             show program's version number and exit
  --no-bind             instance will bind itself to RabbitMQ by name but NOT
                        file type
```



5. Edit the Dockerfile

Update the Dockerfile to install your software dependencies, provide necessary instructions in Dockerfile using the RUN command. You will need to add a line in Dockerfile to switch to the root user (`USER root`) for getting proper permissions. For e.g., to install ImageMagick package using apt-get, add the following commands to Dockerfile:

```
USER root
RUN apt-get update && apt-get install -y imagemagick
```

6. Test the Extractor

You can test your extractor as follows:

```
docker build -t <your_extractor_name> .
docker run -it --link browndog_clowder_1 --link browndog_rabbitmq_1:rabbitmq <your_extractor_name>
```

You should see the following in the terminal. This means that the extractor is running and waiting for messages:

```
INFO    : pyclowder.extractors -  Waiting for messages. To exit press CTRL+C
```

Converters

Here we described the process for taking a working piece of code (an application, library, other service, etc) and deploying it as a Brown Dog converter. In this example, we describe the creation of a converter using the popular image converter tool, ImageMagick.

1. Get Your Code Together

We have developed a template converter. It is a simple image converter that converts between different image formats using ImageMagick tool. Clone the template converter and rename the directory to an appropriate name that reflects the purpose of your converter

```
git clone https://opensource.ncsa.illinois.edu/bitbucket/scm/bd/convertors-template.git
mv convertors-template/ <your_converter_name>
cd <your_converter_name>
```

Rename and edit *ImageMagick_convert.sh* script to wrap your conversion tool. This script file should be named in the format **<alias>_convert.<script_type>**. Here **<alias>** needs to be replaced by the name of the conversion tool with which the converter registers with Polyglot and **<script_type>** needs to be replaced by the extension for the type of script this wrapper is written in. Polyglot currently supports scripts written in Python, Bash, R, AutoHotKey, AutoIT, and Sikuli (e.g. *.py, *.sh, etc.). For the sake of ease of explanation, we will rename the script file as **MyTool_convert.sh**. This script accepts three parameters:

1. Full path to input file
2. Full path to output file (including filename)
3. Full local path to available scratch space (optional)

This script will be used by the Software Server to run the tool and carry out any requested conversions. The example script ImageMagick_convert.sh that uses ImageMagick tool to convert images between different formats is shown below. The conversion script follows a specific header and is written as comments:

1. First line is the shebang line
2. Second line contains the name of the converter followed by the version (if any)
3. Third line refers to the type of the data that it can convert
4. Fourth line contains a comma-separated list of the input file formats accepted by this converter
5. Fifth line contains a comma-separated list of the output file formats that this converter can generate
6. This is followed by the actual code that does conversion.

```
#!/bin/sh
#ImageMagick (v6.5.2)
#image
#bmp, dib, eps, fig, gif, ico, jpg, jpeg, jp2, pcd,
#bmp, dib, eps, gif, jpg, jpeg, jp2, pcd, pdf, pgm,

output_filename=$(basename "$2")
output_format="${output_filename##*.}"

#Output PGM files as ASCII
if [ "$output_format" = "pgm" ]; then
        convert "$1" -compress none "$2"
else
        convert "$1" "$2"
fi
```

2. Edit the Dockerfile

Modify the Dockerfile in the converter directory to replace ImageMagick with MyTool. Specifically change line numbers 11, 15, 16 and 17. You need to also change other fields like maintainer and may need to add instructions to install any specific software required by your converter. For example, you can see instruction to install ImageMagick software in the example Dockerfile:

Dockerfile

```
# Create softwareserver for polyglot.
FROM ncsapolyglot/polyglot:develop
MAINTAINER Rob Kooper <kooper@illinois.edu>

USER root
# - install requirements
# - enable shellscripts to be scanned
# - enable imagemagick conversion by adding to .aliases.txt
RUN apt-get update && apt-get -y install vim nano imagemagick && \
        /bin/sed -i -e 's/^\([^#]*Scripts=\)/#\1/' -e 's/^#\(ShellScripts=\)/\1/' /home/polyglot/polyglot
/SoftwareServer.conf && \
        echo "ImageMagick" > /home/polyglot/polyglot/scripts/sh/.aliases.txt

# copy convert file to scripts/sh folder in container
# this is done to keep cache so you can debug script easily
COPY ImageMagick_convert.sh /home/polyglot/polyglot/scripts/sh/
RUN chown polyglot /home/polyglot/polyglot/scripts/sh/ImageMagick_convert.sh && \
    chmod +x /home/polyglot/polyglot/scripts/sh/ImageMagick_convert.sh

# back to polyglot
CMD ["softwareserver"]
```

Specifically, modify:

```
echo "ImageMagick" > /home/polyglot/polyglot/scripts/sh/.aliases.txt
```

to:

```
echo "MyTool" > /home/polyglot/polyglot/scripts/sh/.aliases.txt
```

modify:

```
COPY ImageMagick_convert.sh /home/polyglot/polyglot/scripts/sh/
```

to:

```
COPY MyTool_convert.sh /home/polyglot/polyglot/scripts/sh/
```

and modify:

```
RUN chown polyglot /home/polyglot/polyglot/scripts/sh/ImageMagick_convert.sh && \
    chmod +x /home/polyglot/polyglot/scripts/sh/ImageMagick_convert.sh
```

to:

```
RUN chown polyglot /home/polyglot/polyglot/scripts/sh/MyTool_convert.sh && \
    chmod +x /home/polyglot/polyglot/scripts/sh/MyTool_convert.sh
```

6. Test the Converter

You can test your converter as follows:

```
docker build –t mytool .
docker run -it --link browndog_rabbitmq_1:rabbitmq mytool
```

You should see the following in the terminal. This means that the converter is running and waiting for messages:

```
Available Software:
  ImageMagick (ImageMagick)
```