Dry Run Instructions

Overview

For dry run system, each team need to complete the following tasks:

- Question generation program
- Question answering program
- Build a docker container which include your programs and all the dependencies
- Upload your docker image and submit your tagged image name to Autolab

The purpose of dry run is to make sure you can correctly build your system as instructions.
The quailty of generated questions and answers will not be graded during dry run phase.

The handout includes:

- **test.sh** which test your system. Please test your container before submission
- **docker** folder, which includes the example files to build a docker container of QA system, you can easily build your own based on this example.

Programs

You must deliver two programs called **ask** and **answer** with the interfaces and properties listed below:

- The programs must have **exactly** the names listed above.
- The programs must be **executable**. If you are using python, please read part 3
- The program must take **exactly** the command line arguments detailed in part 1 and part 2

Both **ask** and **answer** should follow:

- expect input containing **non-ASCII characters in UTF-8 encoding**. Failure to address this will result in your team
  receiving no credit for the competitive portion of the project grade.
- **Debugging information should not be directed to STDOUT**. This will throw off the alignment of your outputs and will result in a dramatically reduced score.

1.Question generation

The ask program must have the following command-line interface:

```
./ask article.txt nquestions
```

Where:

- *article.txt* is a path to an arbitrary plain text file (the document)
- *nquestions* is an integer (the number of questions to be generated).

The program
should output to STDOUT a sequence of questions with each question terminated by a newline character. **Each line should not contain any text other than the question.** If your program cannot generate the requested number of questions, it should not try to buffer the output with empty lines.

2.Question answering

The **answer** program must have the following command-line interface:

```
./answer article.txt questions.txt
```

Where:

- *article.txt* is a path to an arbitrary plain text file (the document)
- *questions.txt* is a path to an arbitrary file of questions (one question per line with no extraneous material).

The output (to STDOUT) should contain
a sequence of answers with each answer terminated by a newline character. If your system cannot generate an answer for a given question, it should output a blank line or a default answer. **Each line should not contain any text other than the hypothesized answer.** Otherwise, your outputs will be penalized for conciseness.

3.Making your Python File Executable on Linux

Step 1: add an appropriate shebang line to beginning of the file. Since the default Python on the servers will be Python 2.7, to run a file with the Python 2.7 interpreter, the first line of your file should be as follows:

```
#!/usr/bin/env python
```

For Python 3, you should instead use:

```
#!/usr/bin/env python3
```

Step 2: change the permissions of the file so that it is executable by all users. At the command line, issue the following command ($ is the command prompt):

```
$ chmod a+x thescript
```

Where thescript is the path to the file whose permissions you want to change. That is it. You should now be able to run the script by typing the following command at the prompt:

```
$ ./thescript
```

Build Docker Containers

A detailed example is included in handout, please checking the files in **docker** folder while reading the following instructions.

Put your system into docker container

What is docker container and why bother using it

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers -
images become containers when they run on Docker Engine.
Available for both Linux and Windows-based applications, containerized software will always run the same,
regardless of the infrastructure.
Containers isolate software from its environment and ensure that it works uniformly despite differences
for instance between development and staging.

Go to (https://docs.docker.com/) for more information,

Step 1: Install docker on your machine

Please refer (https://docs.docker.com/install/)

Step 2: Define a container with Dockerfile

Dockerfile defines what goes on in the environment inside your container.
Access to resources like networking interfaces and disk drives is virtualized inside this environment,
which is isolated from the rest of your system, so you need to map ports to the outside world,
and be specific about what files you want to “copy in” to that environment. However, after doing that,
you can expect that the build of your app defined in this Dockerfile behaves exactly the same wherever it runs.

Here is the example Dockerfile:

```
# Ubuntu Linux as the base image
FROM ubuntu:16.04

# Set UTF-8 encoding
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8

# Install packages, you should modify this based on your program
RUN apt-get -y update && \
    apt-get -y upgrade && \
    apt-get -y install python3-pip python3-dev &&\
    pip3 install spacy && \
    python3 -m spacy download en_core_web_lg
```

```
# Add the files into container, under QA folder, modify this based on your need
ADD ask /QA
ADD answer /QA

# Change the permissions of programs, you may add other command if needed
CMD ["chmod 777 ask"]
CMD ["chmod 777 answer"]

# Set working dir as /QA
WORKDIR /QA
ENTRYPOINT ["/bin/bash", "-c"]
```

The example dockerfile do the following:

- use Ubuntu Linux as its base image(in most case, you don't need to change that)
- Set UTF-8 as encoding (in most case, you don't need to change that)
- Install all the program and dependencies needed for our program, in this example, the python3 is installed,
  since the **ask** and **answer** in this example is written by python3. The package **spacy** is installed, and pre-trianed model en_core_web_lg is downloaded beacause they are dependencies of programs. (you need to change this part based on the dependencies of your programs)
- Copy the files into container. You need to copy all your programs into container. In this example, they are ask and answer. (You may need to change that if you have other program files)
- Change the permissions of programs. (you may add other command if needed)

Step 3: build the container image

Now, everything is ready, not too hard, isn't it? :-), just run command:

```
docker build --tag=${NAME} .
```

${NAME} can be whatever name you want for example

```
docker build --tag=spacemonkey .
```

Use command:

```
docker image ls
```

to check if you successfully build the docker image.

Testing

Run commamds:

```
chmod 777 test.sh && test.sh ${image_name}
```

${image_name} should be your container image you want to test.

if everything is right, the output should be:

```
**************************
Q1
Q2
Q3
**************************
A1
A2
A3
**************************
```

Q1, **Q2** and **Q3** should be three question generated by your program ask,

A1, **A2** and **A3** should be three answers generated by your program answer,

Share your container image with us

At this point, you have finsih building your container image, well done! Just a few more steps to go.

Step 4: Log in with your Docker ID

If you don't have a Docker account, sign up for one. Make note of your username.

Log in to the Docker public registry on your local machine.

```
$ docker login
```

Step 5: Tag the image

Now, put it all together to tag the image. Run `docker tag image` with your username, repository, and tag names so that the image uploads to your desired destination. The syntax of the command is:

```
docker tag image username/repository:tag
```

For example:

```
docker tag spacemonkey tyler/nlpdryrun:version2
```

Use command:

```
docker image ls
```

to check if your newly tagged image. Don't forget to test new image with `test.sh username/repository:tag`

Step 5: Publish the image

Upload your tagged image to the repository:

```
docker push username/repository:tag
```

Once complete, the results of this upload are publicly available.
If you log in to Docker Hub, you see the new image there, with its pull command.

##Submission

Before submission, please create group in autolab:

- step 1: open Project Dry Run on autolab
- step 2: click **Group options** on the left
- step 3: Set Group Name, add partners and create group

Please submit a zip archive containing a single txt file called `docker.txt`.

`docker.txt` should have two line, first line is your published image name `username/repository:tag`, second line is your team name.

Evaluation and Grading

The dry run system will be graded according to the following criteria:

- ask runs on an arbitrary document without errors and generates appropriate output. (50%)
- answer runs on a arbitrary document and a set of questions without errors and generates appropriate output. (50%)

Start early, come to office hours or email TAs if you have problems