PharmaSUG China 2022 - Paper AT-153

Deploying Jupyterhub using Docker

Haiping Yu, dMed

ABSTRACT

Project Jupyter is a game changer in data science and has emerged as a de facto standard. It's a free, open-source, interactive web tool which supports all programming languages. JupyterLab is the latest development environment for notebooks, code, and data, Jupyterhub brings that power to groups of users without burdening the users with installation and maintenance tasks. Docker is popular virtualization methodology that enables you to separate your applications from your infrastructure so you can deliver and scale software quickly. Today we are going to introduce how to deploy JupyterHub using Docker, to make the computing platform scalable and suitable for both small and large teams.

INTRODUCTION

In this article, we will introduce Jupyter project and its components, the virtualization technology Docker, and focus on the details of deploying Jupyterhub using Docker.

WHAT IS JUPYTER

Jupyter project

Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license.

Jupyter notebook

The Jupyter Notebook is the original web application for creating and sharing computational documents. It enables users to author notebook documents that include live code, interactive widgets, plots, narrative text, equations, images and video. It offers a simple, streamlined, document-centric experience.

JupyterLab

JupyterLab is a next-generation web-based user interface. It transforms the classic notebook interface in a full featured IDE, with multiple split views, file explorer, Jupyter notebooks, and much more, in a single browser window.

JupyterHub

Whereas Jupyter notebooks are meant to run on a personal computer. JupyterHub is the solution to serve Jupyter notebook for multiple users. It can be used in a class of students, a corporate data science group or scientific research group. It is a multi-user Hub that spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.

A JupyterHub deployment has

- a Hub (tornado process) that is the heart of JupyterHub
- a configurable http proxy (node-http-proxy) that receives the requests from the client's browser
- multiple single-user Jupyter notebook servers (Python/IPython/tornado) that are monitored by Spawners
- an authentication class that manages how users can access the system

Besides these central pieces, you can add optional configurations through a config.py file and manage users kernels on an admin panel. A simplification of the whole system can be seen in the figure below:

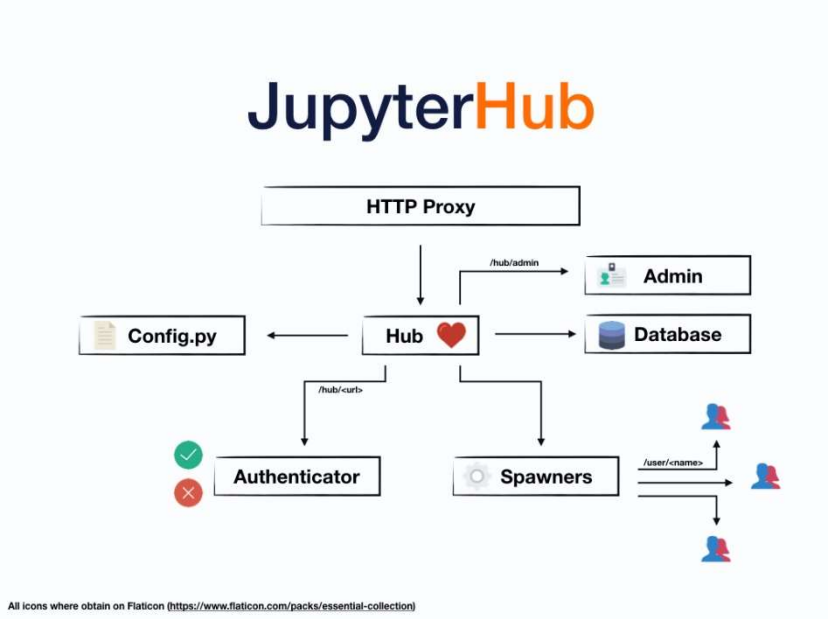

Figure 1. JupyterHub Architecture

WHAT IS DOCKER

Docker is a kind of virtualization technology that makes it easy for us to develop and deploy applications inside of neatly packaged virtual containerized environments. Meaning applications run the same, no matter where they are of what machine they are running. Docker containers can be deployed to just about any machine without any compatibility issues, so your software stays system agnostic, making software simpler to use, less work to develop, and easy to maintain and deploy. These containers running on your computer or server act like micro computers, with very specific jobs, each with their own operating system and their own isolated CPU, Memory, and network resources. And because of this, they can be easily added, removed, stopped and started again without affecting each other of the host machine. Containers usually run one specific task such as a PostgreSQL database or a GitLab application, and then network together, and potentially scaled.

HOW TO DEPLOY JUPYTERHUB USING DOCKER

DIFFERENT APPROACHES

There two official distributions of JupyterHub deployment.

- The Littlest JupyterHub deployment is on a single server with no virtualization or containerization technology. It is perfect for a small amount of users (0-100).
- the Zero to JupyterHub with Kubernetes deployment targets very large clouds with dynamic amount of servers, managed through Kubernetes. It is adapted for very large organizations with up to thousands of users.

We took the middle approach, a medium-scale deployment, suitable for up to hundreds of users. We choose virtualization, but no cluster servers, deploying JupyterHub using Docker on one server.

THE FOLDER STRUCTURE

To simplify the management, and automatizing the deployment, we will use Docker Compose. All the configurations will be kept in a single folder. The following shows the folder structure.

```
./
|-- jupyterhub/
|   |-- Dockerfile
|   |-- jupyterhub_config.py
|-- jupyterlab/
|   |-- Dockerfile
|   |-- conda-activate.sh
|-- .env
|-- docker-compose.yml
```

The main configuration file docker-compose.yml defines all containers and associated volumes, each of the two subfolders has the configuration files for each container. It will look like the following. You can visit the official Compose file version 3 reference webpage to learn more details and options.

```
version: '3'

services:
  # Configuration for Proxy+Hub
  jupyterhub:
    ……

  # Configuration for the single-user servers
  jupyterlab:
    ……

volumes:
  ……
```

We now explain the configuration of each service, and the contents of each additional file in detail.

THE JUPYTERHUB

docker-compose.yml

In the Docker Compose file docker-compose.yml, the jupyterhub section defines where to build and how to configure the JupyterHub container.

```
  jupyterhub:
    build: jupyterhub           # Build the container from this folder.
    image: jupyterhub_img       # Specify the image to start the container from.
    container_name: jupyterhub  # Specify the custom container name.
    volumes:                    # Mount host paths or named volumes.
```

```
      - /var/run/docker.sock:/var/run/docker.sock
      - jupyterhub_data:/srv/jupyterhub     # Hub data persistence
    environment:                # Env variables passed to the Hub process.
      DOCKER_JUPYTER_IMAGE: jupyterlab_img
      DOCKER_NETWORK_NAME: ${COMPOSE_PROJECT_NAME}_default
      HUB_IP: jupyterhub
    restart: on-failure         # The restart policy.
    ports:
     - "8000:8000"              # Expose the ports.
```

environment variables

In this Docker Compose file docker-compose.yml, we set some environment variables for the Hub process, they will be used in the Hub configuration file jupyterhub_config.py.

- DOCKER_JUPYTER_IMAGE is the name of the Docker image for the single-user servers; this must match the image configured in the jupyterlab section.
- DOCKER_NETWORK_NAME is the name of the Docker network used by the services. We pass another environment variable COMPOSE_PROJECT_NAME by adding a .env file next to the docker-compose.yml.

```
COMPOSE_PROJECT_NAME=jupyterhub
```

- HUB_IP is the IP address of the Hub service within the docker network.

Dockerfile

In the JupyterHub Dockerfile, it is important to explicitly specify the version of the image that will be pulled from the Docker Hub, and make sure it's compatible with other modules.

```
FROM jupyterhub/jupyterhub:1.3.0
COPY jupyterhub_config.py .
RUN pip install \
    dockerspawner==12.0.0 \
    jupyterhub-firstuseauthenticator==0.14.1 \
    jupyterhub-idle-culler==1.1
```

jupyterhub_config.py

Next to Dockerfile, we have jupyterhub_config.py to configure the Hub. This plain Python file contains many different sections. You can visit the official JupyterHub documentation website to find more options.

DockerSpawner

We start with the configuration of the spawner, we use the class DockerSpawner to spawn single-user servers in a separate Docker container. We use here the environment variables that we have set in docker-compose.yml.

```
c.JupyterHub.spawner_class = 'dockerspawner.DockerSpawner'
```

```
c.DockerSpawner.image = os.environ['DOCKER_JUPYTER_IMAGE']
c.DockerSpawner.network_name = os.environ['DOCKER_NETWORK_NAME']
c.JupyterHub.hub_ip = os.environ['HUB_IP']
```

Idle handling

To save the resource, we may want to stop the single-user servers after a certain amount of idle time. Following this example, we register a JupyterHub service like this:

```
c.JupyterHub.services = [
    {
        'name': 'idle-culler',
        'admin': True,
        'command': [
            sys.executable,
            '-m', 'jupyterhub_idle_culler',
            '--timeout=1800'
        ],
    }
]
```

Authentication

For a multiple-user application, it's necessary to configure the authentication method. The JupyterHub ships with the default PAM-based Authenticator, here we choose a simplified authenticator called firstuseauthenticator, users can just pick a username and password on their first login and get to work! There are many other official and third-party authenticators, such oauthenticator and ldapauthenticator. You can also write a custom authenticator by your own. Visit the official JupyterHub Authenticators webpage for more details.

```
from firstuseauthenticator import FirstUseAuthenticator
c.JupyterHub.authenticator_class = 'firstuseauthenticator.FirstUseAuthenticator'
c.Authenticator.admin_users = { 'admin' }
```

Data persistence

Using Docker volumes, we can make storage permanent. There are two kinds of data need to be stored, the Hub data, containing information on administrators and existing users, and the user's data for the single-user servers. The Hub data persistence is already defined in the Docker Compose file docker-compose.yml. To persist the user's data, we add these lines, even if we remove the images and containers, the user data will not be lost. In addition, users have both the private and shared workspaces.

```
notebook_dir = os.environ.get('DOCKER_NOTEBOOK_DIR') or '/home/jovyan'
c.DockerSpawner.notebook_dir = notebook_dir
c.DockerSpawner.volumes = {
'jupyterhub-user-{username}': notebook_dir ,
'jupyterhub-shared': '/home/jovyan/shared'
}
```

Other configurations

If you want to user JupyterLab as the default interface for your JupyterHub users, configure the Hub by adding this line to jupyterhub-config.py:

```
c.Spawner.default_url = '/lab'
```

We can also set the upper limit of CPU and memory a single-user notebook server is allowed to use.

```
c.Spawner.cpu_limit = 1
c.Spawner.mem_limit = '2G'
```

To allow the admins have permission to log in as other users on their respective machines, for debugging, JupyterHub.admin_access is set to True.

```

c.JupyterHub.admin_access = True
```

THE JUPYTERLAB

docker-compose.yml

In the Docker Compose file docker-compose.yml, the jupyterhub section defines where to build and how to configure the single-user server, the JupyterLab container.

```
  jupyterlab:
    build: jupyterlab
    image: jupyterlab_img
    network_mode: none
    command: echo
```

Dockerfile

In the JupyterLab Dockerfile, we also need to explicitly specify the version of the image that will be pulled from the Docker Hub. The Jupyter Docker Stacks website can help you to select the image.

```
FROM jupyter/scipy-notebook:4d9c9bd9ced0
```

You can also customize the image.

- Add R packages

```

RUN conda install --quiet --yes \
    'r-base=3.4.1' \
    'r-irkernel=0.8*' && \
    conda clean -tipsy && \
    fix-permissions $CONDA_DIR
```

- Add Python packages

```
RUN pip install \
```

```
    sas7bdat==2.2.3 \
    fix-permissions $CONDA_DIR
```

- Set time zone

```
ENV TZ=Asia/Shanghai
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

INSTALL DOCKER AND DOCKER COMPOSE

The operation system we used is the CentOS, to install the latest version of Docker Engine, containerd, and Docker Compose, we just need to type the one command line. You can visit the official Install the Docker Engine webpage to get more details about the installation instructions.

```
sudo yum install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

RUN DOCKER COMPOSE

You are now ready to test your JupyterHub server. Run the following commands.

```
docker-compose build
docker-compose up
```

If all configurations are proper, you can visit the website http://[IPADDRESS]:8000/ , test the authentication, launch the single -user server, and play with some notebooks. When everything is ok, you can stop the server by type Ctrl+C.

Once you are ready to move to production, run the following command.

```
docker-compose up -d
```

CONCLUSION

We have been running the setup for years, and it works very well, with the digitalization of clinical trials, we thought it's the time for us to embrace Data Science and DevOps.

REFERENCES

Luca De Feo. "Deploying a containerized JupyterHub server with Docker." 17 Oct 2018. https://opendreamkit.org/2018/10/17/jupyterhub-docker/ .