# TA Section Week 4

9/27/2018 8-9PM

Zhenyu Zhao

DevOps Engineer

Infrastructure Technology Services

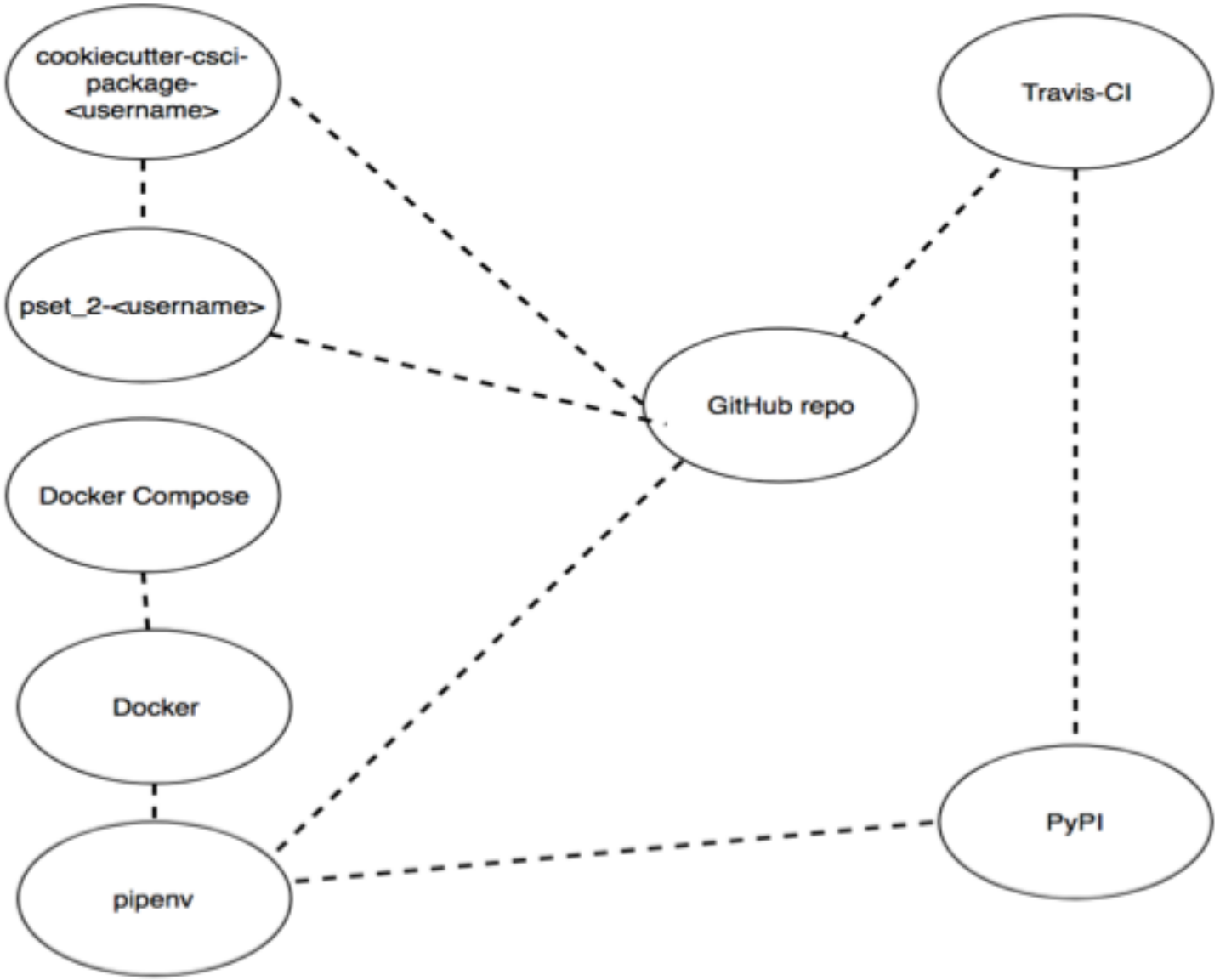
Harvard University IT

Email: Zhenyu_zhao@Harvard.edu

# Agenda

- ❑Pset2 Overview
- ❑Docker Compose
- ❑Docker
- ❑Travis CI
- ❑Environment variables

# Pset2 Overview



# Docker Compose #1

- UsinDocker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration
- g Compose is basically a three-step process:
  - ✓ Define your app's environment in Dockerfile so it can be reproduced anywhere.
  - ✓ Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  - ✓ Run "docker-compose build" to build Docker images
  - ✓ Run "docker-compose up" to start and run your entire app in a Docker container.

# Docker Compose #2

- docker-compose.yml
  - ✓ The docker-compose.yml file is a YAML file defining services, networks and volumes.
  - ✓ A service definition contains configuration that is applied to each container started for that service, much like passing command-line parameters to ”docker container create”
  - ✓ A volume definition contains configuration that is applied to each container started for that volume, much like passing command-line
  - ✓ A network definition contains configuration that is applied to each container started for that volume, much like passing command-line parameters to ”docker network create”
- ✓ An sample docker-compose.yml file

```
version: '3'
services:
  app:
    build:
      context: .
      args:
        - CI_USER_TOKEN=${CI_USER_TOKEN}
    volumes:
      - .:/app
    environment:
      - PYTHON
```

# Docker #1

- Docker refers to a popular innovative container technology from Docker CE
- Docker images vs. Docker container
  - ✓ Docker image
  - ✓ Docker container
- Dockerfile: a text file containing a set of instructions that specify what environment to use and which commands to run.
  - ✓ FROM: initializes a new build stage and sets the base Image for subsequent instructions. This is MUST
  - ✓ ARG: comes before or after FROM. An ARG declared before a FROM is outside of a build stage, so it can't be used in any instruction after a FROM. An ARG declared before a FROM can.

```
ARG VERSION=latest
FROM busybox:$VERSION
ARG VERSION
RUN echo $VERSION > image_version
```

  - ✓ RUN: execute any commands in a new layer on top of the current image and commit the results.
  - ✓ ENV: declare environment variable
  - ✓ CMD: provide defaults for an executing container.

# Docker #2

- ✓ WORKDIR: sets the working directory within container's file system for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile. If the WORKDIR doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.

✓ A sample Dockerfile

```
FROM python:3.6 AS base

ARG CI_USER_TOKEN
RUN echo "machine github.com\n login $CI_USER_TOKEN\n" > ~/.netrc

RUN pip install pipenv

ENV \
  PYTHONFAULTHANDLER=1 \
  PYTHONUNBUFFERED=1 \
  PYTHONHASHSEED=random \
  PIP_NO_CACHE_DIR=off \
  PIP_DISABLE_PIP_VERSION_CHECK=on \
  PIP_DEFAULT_TIMEOUT=100 \
  PIPENV_HIDE_EMOJIS=true \
  PIPENV_COLORBLIND=true \
  PIPENV_NOSPIN=true \
  PYTHONPATH="/app:${PYTHONPATH}"

WORKDIR /build

RUN pipenv install

COPY Pipfile .
COPY Pipfile.lock .

RUN pipenv install --system --deploy --ignore-pipfile --dev
WORKDIR /app
```

# Travis CI #1

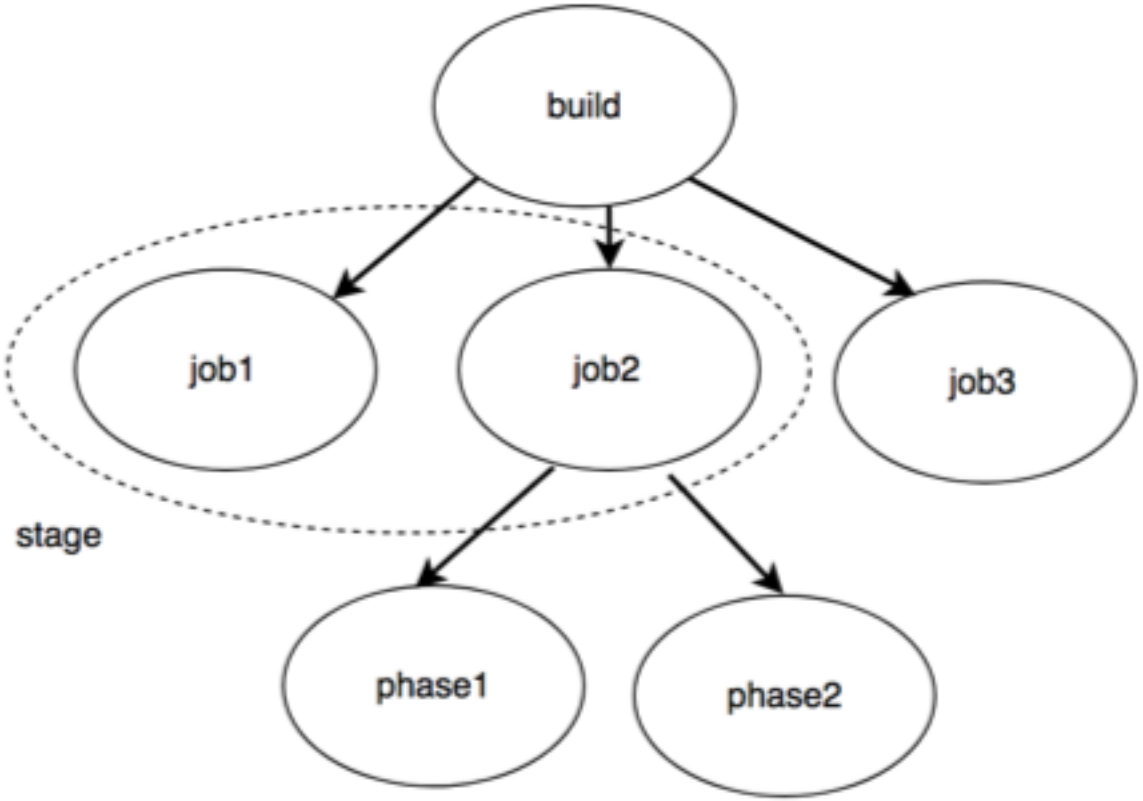
- Travis CI is hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.
- Public projects may be tested at no charge via travis-ci.org. Private projects may be tested at travis-ci.com on a fee basis.
- How does Travis CI work with GitHub?
  - ✓ Travis CI is configured by adding a text file named .travis.yml, which is a YAML format text file, to the root directory of the repository.
  - ✓ When Travis CI has been activated for a given repository, GitHub will notify it whenever new commits are pushed to that repository or a pull request is submitted.
  - ✓ It can also be configured to only run for specific branches, or branches whose names match a specific pattern. Travis CI will then check out the relevant branch and run the commands specified in .travis.yml, which usually build the software and run any automated tests.
  - ✓ When that process has completed, Travis notifies the developer(s)

# Travis CI #2

- Travis CI provides a default build environment and a default set of steps for each programming language. But you customize the build environment by customizing .travis.yml
- A build on Travis CI is made up of two steps:
  - ✓ install: install any dependencies required
  - ✓ script: run the build script
- A build lifecycle in .travis.yml
  - ✓ apt install (optional)
  - ✓ cache components (optional)
  - ✓ before_install (optional)
  - ✓ *install*
  - ✓ before_scritp (optional)
  - ✓ *script*
  - ✓ before cache (optional)
  - ✓ after_success or after_failure (optional)
  - ✓ deploy (optional)
  - ✓ after_deploye (optional)
  - ✓ after_script (optional)

# Travis CI #3

- Build, job, phase, and stage



# Travis CI #3

- Other important options in .travis.yml
  - ✓ sudo: launch a Ubuntu VM
  - ✓ services: specify docker service to be installed by the build environment

```
sudo: required
language: minimal
services:
  - docker
```

  - ✓ jobs: an automated process that clones your repository into a virtual environment and then carries out a series of phases such as compiling your code, running tests, etc. A job fails if the return code of the script phase is non zero.

- A sample .travis.yml

```
sudo: required
language: minimal
services:
- docker
install: docker-compose build
jobs:
  include:
  - stage: test
    if: branch = develop
    script: ./drun_app pytest
  - stage: deploy
    if: branch = master
    script: ./drun_app python main.py
```

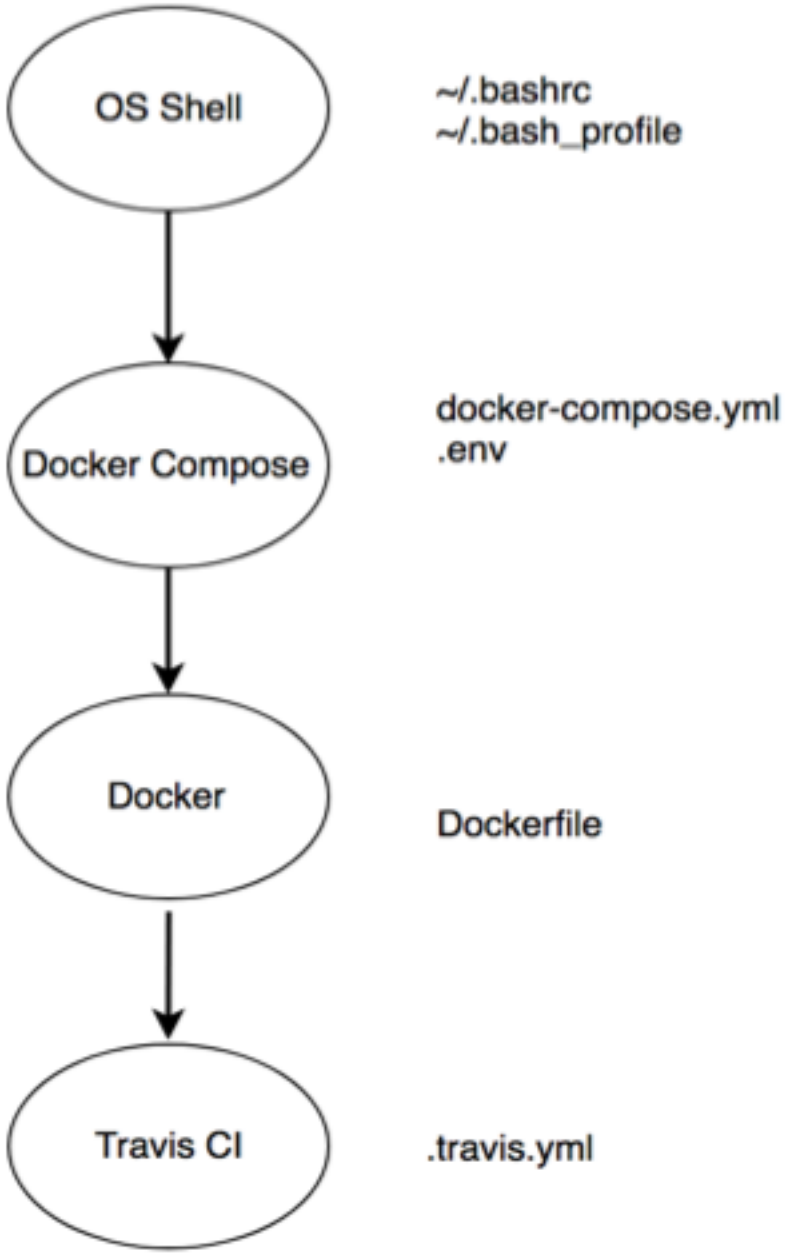
```
env:
  global:
    secure: C04ZnzpOpGoRDAgpnAfMgVxBmrroXx9t70xyvHFEmi9D524VZqSPZicZDOea8NXatZP
xCw4lOBCgAyu+7u/tkZ+KUqKXD6JWl3nOqZ0ZB183X/MUrwZPmM0qctFNowUDKIdogqGRWYJk/h8u+G
lYg2EcrhXJMjaHg9ddxOHRUf5t94AxEcNangkzld8msQi0ecxd1iyXekSbcD0iOc4HnDdeErGIXCm4+
YVOnTUCAR45Vwjl4JjIsG7rj1Ps2QEeMfq6zLDtRDLetw7izBgzgcU71cXBM6kuCWV06b7DP2DRXnty
iTUhdoF98Yb3vqg=
```

# Environment variables #1

- Three ways to set environment variables in a build environment
  - ✓ ~/.bash or ~/.bash_profile
  - ✓ .env (in the root directory of the project directory)
  - ✓ docker-compose.override.yml

# Environment variables #1

- How environment variable values are transitioned



# Environment variables #2

- OS Shell environment variable
  - ✓ ~/.bashrc
  - ✓ ~/.bash_profile

```
# GitHub zhenyuzhao account personal access toke
CI_USER_TOKEN=60f51a4978cc9005742acb5689a66ee3dda44be4
```

Once set, run the following command to make it take effect in the current shell:

$ source ~/.bashrc

$ source ~/.bash_profile

- Docker Compose
  - ✓ docker-compose.yml

```
version: '3'
services:
  app:
    build:
      context: .
      args:
        - CI_USER_TOKEN=${CI_USER_TOKEN}
    volumes:
      - .:/app
    environment:
      - PYTHON
```

# Environment variables #3

- Docker
  - ✓ Dockerfile

```
ARG CI_USER_TOKEN
RUN echo "machine github.com\n login $CI_USER_TOKEN\n" > ~/.netrc
```

- Travis CI
  - ✓ You need to encrypt the token and put it to .travis.yml

$ travis encrypt -r csci-e-29/pset_utils-zhenyuzhao CI_USER_TOKEN=60f51a4978cc9005742acb5689a66ee3dda44be4 --add

The result is:

```
script: ./uirun_app pytest
env:
  global:
    secure: C04ZnzpOpGoRDAgpnAfMgVxBmrroXx9t70xyvHFEmi9D524VZqSPZicZDOea8NXatZPZuTYFL/pqy7aIPL053VgmcpYavNPsiDMfuAGUCUWAOdkUsCnvtR3vH3F6n9hqsLgGoj4vmXgjKDfDMnvlpfAssefUSl
xCw4lOBCgAyu+7u/tkZ+KUqKXD6JWl3nOqZ0ZB183X/MUrwZPmM0qctFNowUDKIdogqGRWYJk/h8u+GiRNOmSZXg7WbdML8xmastVpskOfySm4oesNsthl4y0Wg18o8PRFI/BFdZXabg8kEX3eLVs+DqlY8CL6yS7L8Kt1oYrp
lYg2EcrhXJMjaHg9ddxOHRUf5t94AxEcNangkzld8msQi0ecxd1iyXekSbcD0iOc4HnDdeErGIXCm4+D0Ubn1LSPP2T+OOV0tui+bA68iwsBNRIfh0ylIFPwxkNrEggO7iyIfqH/jOIsFFD0FkdOrAvTCBZhJhgdaJ40hEsTFc
YVOnTUCAR45Vwjl4JjIsG7rj1Ps2QEeMfq6zLDtRDLetw7izBgzgcU71cXBM6kuCWV06b7DP2DRXntyxy+JxN1vGfqYHwWO2OyWsGDI5zHjwBGugZ/TrU88zPYc8Bp60PZSKWiBUlbj1GZ5nU2ZHLRv6w2x3w5GL7nOa0B5QxY
iTUhdoF98Yb3vqg=
```