Workflow automation using Docker Swarm and GitLab CI

@ Flanders Institute for Biomechanical Experimentation

Johan Philips - NL-RSE - November 20th, 2019

Find slides at https://u0052546.pages.mech.kuleuven.be/presentations/rse/ (non-IE browser)
A bit of context...
What is FIBEr?

Mechanical properties characterization of biological tissues and biomaterials
FIBER team
FIBEr statistics (May 2019)

54 researchers gained access to FIBEr Cloud Services
1565 labels printed with FIBEr Labeler
1233 samples registered in FIBEr Database
202 experiments registered via FIBEr Dashboard or FIBEr Uploader
368 datasets packed and shipped to Data Center
328.27 GB safely stored at ICTS data center
21.4 TB temporary kept on FIBEr Buffer
The need for workflow automation

**Traceability** - Activities and manipulation with samples are logged.

**Safe Storage** - Experimental data is automatically uploaded.

**Error Resilience** - Automated data collection and validation reduces human error.

**Ease of Use** - Intuitive guided workflows help researchers during experimentation.
What do we offer?
FIBEr frontends for everyday lab workflows

Software development to support FIBEr researchers
Reused in already five other labs!
Automation support for other research workflows

GitLab CI for version control and auto-build of LaTeX publications

```
# use docker image with latex preinstalled
image: registry.gitlab.mech.kuleuven.be/gitlab/latex:master

variables:
  # The directory containing your tex files
  PATH_TO_TEX_FILES: src

build:
  script:
    - cd $PATH_TO_TEX_FILES
    - latexmk -pdf
  artifacts:
    paths:
      - $PATH_TO_TEX_FILES/*.pdf

  # Make sure that build job is only run by Gitlab runners tagged for latex
  tags:
    - latex
```
Automation support for other research workflows (2)

GitLab Pages for automated web pages for lectures, research, staff info

```
pages:
  image: python:alpine
before_script:
  - pip install mkdocs
  - pip install mkdocs-bootstrap
script:
  - mkdir -p docs
  - cp README.md docs/index.md
  - cp lecture-*.md docs/
  - cp -R img/ docs/
  - mv theme/ custom_theme/
  - mkdocs build -d public
artifacts:
  paths:
    - public
only:
  - master
tags:
  - pages
```
Automation support for other research workflows (3)
Custom GitLab CI pipelines to improve reproducibility
GitLab Issue board for 'support tickets' and software project management
So what is behind the scenes...?
DevOps@MECH & MECH Cloud
In-house cloud infrastructure to support research labs @MECH - KU Leuven

Enabling secure data management, application deployment, data processing, simulations

Set up and support by 1-2 RSEs (yes, that includes me :-))

Servicing already > 10 research groups
Backed by Docker Swarm and GitLab CI/CD!
FIBEr setup
Hardware backend

5 CoreOS nodes on Intel Xeon E5-2640 v4, 25M Cache, 2.40 GHz
480 GB SSD, 192 GB RAM, 1 TB NFS
Provisioned with XenCenter & Cloud Config

coreos:
units:
- name: docker.service
  command: start
  enable: true
  # Hypervisor Linux Guest Agent
- name: xe-linux-distribution.service
  command: start
  content:
  [Unit]
  Description=Hypervisor Linux Guest Agent
  After=docker.service
  [Service]
  ExecStartPre=/media/configdrive/agent/xe-linux-distribution /var/cache/xe-linux-distribution
  ExecStart=/media/configdrive/agent/xe-daemon
Docker Swarm configuration

$ docker swarm init
Swarm initialized: current node (ip9w0ds0lius3eryxuj3mluus) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-3e0hh0j5t4yjg209f4g5qpowbsczhv2dea9alay218787cf-2h41y330d0j917cvzw30j5x9 10.112.72.1

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

$ docker node ls

<table>
<thead>
<tr>
<th>ID</th>
<th>HOSTNAME</th>
<th>STATUS</th>
<th>AVAILABILITY</th>
<th>MANAGER STATUS</th>
<th>ENGINE VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip9w0ds0lius3eryxuj3mluus *</td>
<td>node1</td>
<td>Ready</td>
<td>Active</td>
<td>Leader</td>
<td>18.06.3-ce</td>
</tr>
<tr>
<td>wceftxg05cac28fsa1x28752r</td>
<td>node2</td>
<td>Ready</td>
<td>Active</td>
<td>Reachable</td>
<td>18.06.3-ce</td>
</tr>
<tr>
<td>760rkmbhgfqyqyqzxtxurr1bv</td>
<td>node3</td>
<td>Ready</td>
<td>Active</td>
<td>Reachable</td>
<td>18.06.3-ce</td>
</tr>
<tr>
<td>tnmxgbkexqaecckzzvew1bee</td>
<td>node4</td>
<td>Ready</td>
<td>Active</td>
<td>Reachable</td>
<td>18.06.3-ce</td>
</tr>
<tr>
<td>7cfpcywjc5v3wybjexwgj3qfk</td>
<td>node5</td>
<td>Ready</td>
<td>Active</td>
<td>Reachable</td>
<td>18.06.3-ce</td>
</tr>
</tbody>
</table>

Docker daemon socket TLS protection: [https://docs.docker.com/engine/security/](https://docs.docker.com/engine/security/)
Docker Swarm Integration with GitLab CI/CD workflow

Declarative specification of GitLab CI pipeline

Source: https://about.gitlab.com/product/continuous-integration/abay
GitLab CI: the basics for Docker Swarm integration

```yaml
image: docker:latest

variables:
  DOCKER_DRIVER: overlay2

stages:
  - build
  - review
  - staging
  - backup
  - production

before_script:
  - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" "$CI_REGISTRY"

# Store Docker Swarm TLS certificates
  - mkdir -p ./docker
    # $USER is $CI_REGISTRY_USER
    - echo "$CLUSTER_CA_CERT" > ~/.docker/ca.pem
    - echo "$CLUSTER_CLIENT_CERT" > ~/.docker/cert.pem
    - echo "$CLUSTER_CLIENT_KEY" > ~/.docker/key.pem

after_script:
  # Logout GitLab Container Registry to remove credentials from Runner
  - docker logout "$CI_REGISTRY"
```
GitLab CI: templates for Docker Stack deployment

```plaintext
.deploy-stack:
  script:
    # Truncate stack name to avoid exceeding 63 char length of docker object names
    # Usually not a problem for production and staging stack, but review apps
    # can potentially create long names
    - export APP_STACK_NAME=${APP_STACK_NAME:0:50}
    - export DOCKER_HOST=${CLUSTER_DOCKER_HOST}
    - export DOCKER_TLS_VERIFY=1
    - docker stack deploy --Compose-file docker-compose.yml
    - ${APP_STACK_FILE}
    tags:
      - docker
  except:
    - schedules

.remove-stack:
  script:
    # Truncate stack name to avoid exceeding 63 char length of docker object names
    # Usually not a problem for production and staging stack, but review apps
    # can potentially create long names. this should be the same length as
    # used in deploy-stack job!
    - export APP_STACK_NAME=${APP_STACK_NAME:0:50}
    - export DOCKER_HOST=${CLUSTER_DOCKER_HOST}
    - export DOCKER_TLS_VERIFY=1
    - docker stack rm ${APP_STACK_NAME}
  when: manual
  tags:
    - docker
  except:
    - schedules
```
GitLab CI: template for MongoDB backup

```bash
. mongodump:

  script:
  # Truncate stack name to avoid exceeding 63 char length of docker object names
  # Usually not a problem for production and staging stack, but review apps
  # can potentially create long names. This should be the same length as
  # used in deploy-stack job!
  - export APP_STACK_NAME=${APP_STACK_NAME:0:50}
  - export DOCKER_HOST=${CLUSTER_DOCKER_HOST}
  - export DOCKER_TLS_VERIFY=1
  - export MONGODUMP_CMD="mkdir -p $MONGODB_BACKUP_DIR; mongodump --username $MONGO_INITDB_DATABASE_USERNAME --password $MONGO_INITDB_DATABASE_PASSWORD --authenticationDatabase $MONGO_INITDB_DATABASE --db $MONGO_INITDB_DATABASE --gzip --archive="$MONGODB_BACKUP_DIR/$MONGO_INITDB_DATABASE-$date +%Y%m%d%H%M.gz""
  - export MONGODB_TASK_ID=`docker service ps --no-trunc |${APP_STACK_NAME} |grep |${APP_MONGODB_SERVICE} | (read ID OTHER; if [ $? -eq 0 ]; then echo $ID; fi)`
  - docker run -v /var/run/docker.sock:/var/run/docker.sock --rm datagridsys/skopos-plugin-swarm-exec task-exec $MONGODB_TASK_ID /bin/bash -c "$MONGODB_DUMP_CMD"

tags:
  - docker
```
GitLab CI: Docker Image integration via Container Registry and deployment environments

```yaml
# Build images from project source and push them to GitLab Container Registry
build-image:
  stage: build
  script:
    - echo "Using image $CI_REGISTRY_IMAGE with tag $CI_COMMIT_REF_NAME"
    # Try to pull image from the registry for use as cache
    - docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME || true
    # Build the image
    - docker build --pull -t $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
    # Push freshly built image
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
  except:
    - tags
    - schedules
  tags:
    - docker

# Deploy production
deploy-production:
  extends: .deploy-stack
  variables:
    - APP_STACK_NAME: $CI_PROJECT_PATH_SLUG
    - APP_STACK_FILE: docker-compose.prod.yml
    - APP_IMAGE: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
    - APP_IMAGE_ID: $CI_REGISTRY_IMAGE_ID
    - APP_IMAGE_DIGEST: $CI_REGISTRY_IMAGE_DIGEST
  stage: production
  environment:
    name: production
    url: https://$CI_REGISTRY_IMAGE:$CI_REGISTRY_REGISTRY_ID
    on_stop: stop-production
  when: manual
    - master

# Stop production
stop-production:
  extends: .remove-stack
  stage: production
  environment:
    name: production
    url: https://$CI_REGISTRY_IMAGE:$CI_REGISTRY_REGISTRY_ID
    on_stop: stop-production
  when: manual
    - master

# Backup production
backup-production:
  stage: backup
  environment:
    name: production
    url: $CI_REGISTRY_IMAGE
    on_stop: stop-production
  when: manual
    - master
```

```
# Deploy staging
deploy-staging:
  extends: .deploy-stack
  variables:
    - APP_STACK_NAME: $CI_PROJECT_PATH_SLUG
    - APP_STACK_FILE: docker-compose.staging.yml
    - APP_IMAGE: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
    - APP_IMAGE_ID: $CI_REGISTRY_IMAGE_ID
    - APP_IMAGE_DIGEST: $CI_REGISTRY_IMAGE_DIGEST
  stage: staging
  environment:
    name: staging
    url: https://$CI_REGISTRY_IMAGE:$CI_REGISTRY_REGISTRY_ID
    on_stop: stop-staging
  when: manual
    - master
    - staging
    - deploy:
        - before:
            - always
  before:
    - deploy:
        - always
```
image: docker:latest

variables:
  DOCKER_DRIVER: overlay2

stages:
  - build
  - review
  - staging
  - backup
  - production

before_script:
  - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  # Store Docker Swarm TLS certificates
Docker Stacks

Declarative specification of Docker elements
E.g. HTTP reverse proxy and load balancer Traefik:

```yaml
version: "3.3"

services:
  traefik:
    image: traefik:alpine
    command: --web
    ports:
      - "80:80"
      - "8080:8080"
      - "443:443"
    volumes:
      - traefik_logs:/logs
      - /var/run/docker.sock:/var/run/docker.sock
    #labels:
    #  - "traefik enable=false"
```
Lessons learned using GitLab CI/CD and Docker Swarm in research...
The Good...

Declarative workflows combined with version control!
Automated deployment of various research workflows
GitLab CI templating allows you to easily reuse and extend
GitLab is a great research tool (software PM, version control, CI/CD, automation, ...)!  
Greatly improved research software and research data management
The Bad...

High learning curve from rapid prototyping to production
Yet Another Management tool for researchers to learn
Research is diverse, so difficult to develop generic tooling
... and the Ugly!

Docker Swarm / CoreOS combo not reliable...
Docker storage management is messy and requires frequent manual clean up
Discipline is required by researchers to optimally improve research reproducibility
Questions?

Source: Pixabay