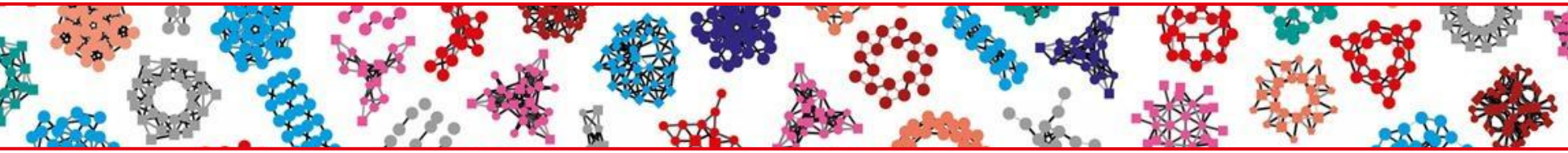
Swiss Institute of Bioinformatics

Beyond the usual Docker tutorial

Web apps & CI

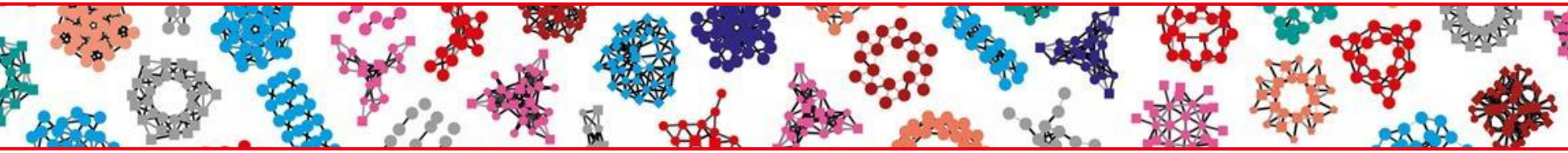
Overview



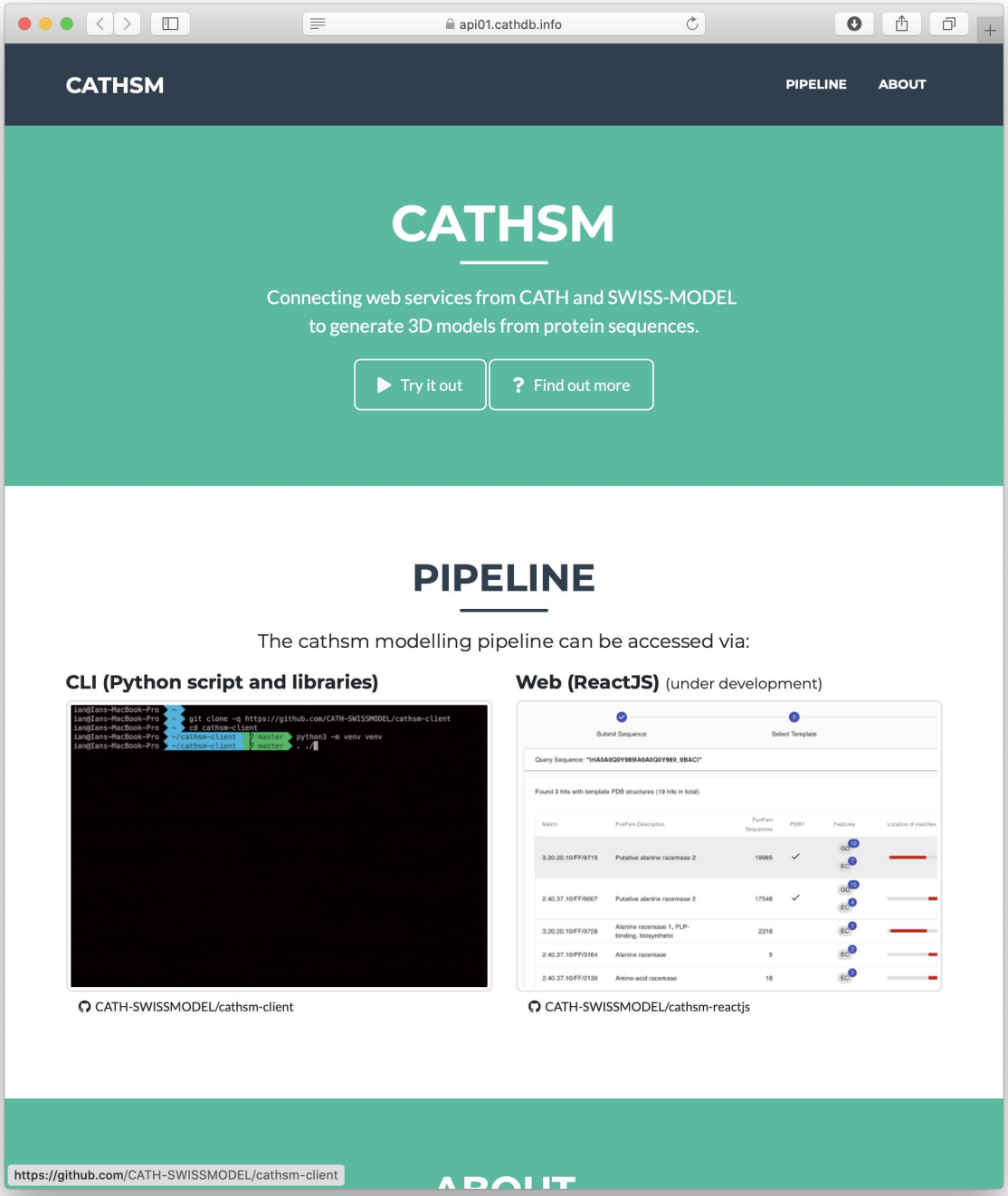
01 Example Application

02 Container Orchestration

03 CI / CD



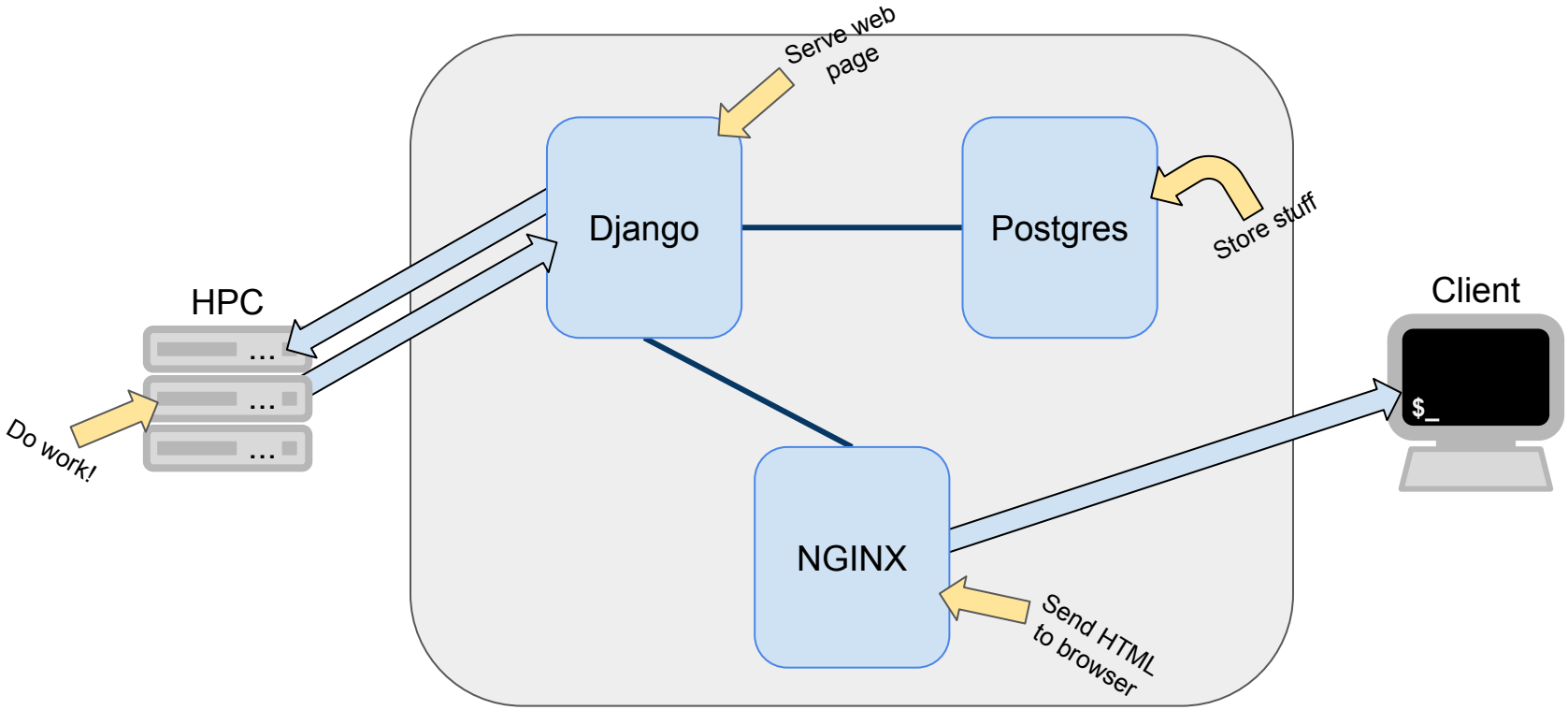
Example Application


api01.cathdb.info
CATHSM
PIPELINE
ABOUT
CATHSM
Connecting web services from CATH and SWISS-MODEL
to generate 3D models from protein sequences.
Try it out
Find out more
PIPELINE
The cathsm modelling pipeline can be accessed via:
CLI (Python script and libraries)
Web (ReactJS) (under development)
CATH-SWISSMODEL/cathsm-client
CATH-SWISSMODEL/cathsm-reactjs
https://github.com/CATH-SWISSMODEL/cathsm-client

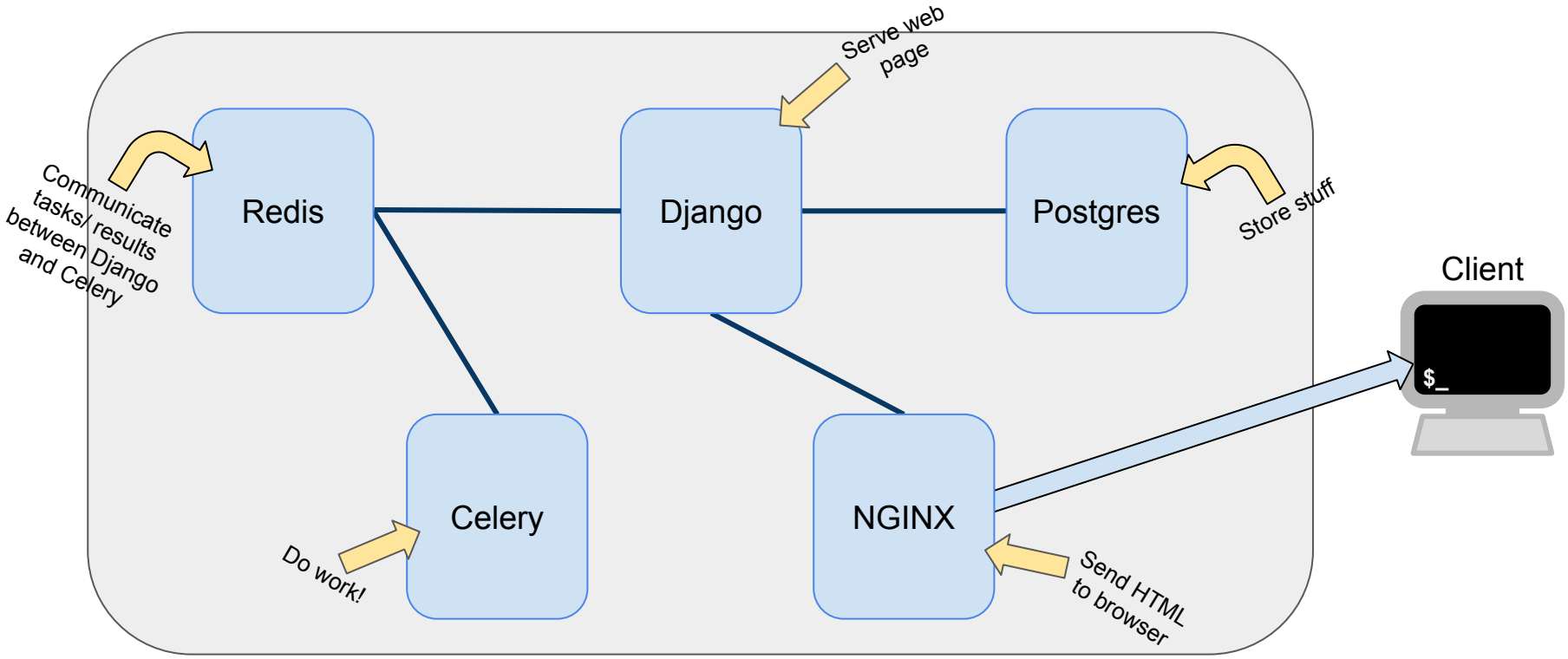
Cath-SM Server

- Collaboration between CATH (cathdb.info) and SWISS-MODEL (swissmodel.expasy.org)
- Implements a new protein modelling pipeline
  - CATH provides the sequence search
  - SWISS-MODEL provides the modelling engine
  - CATH-SM Server provides communication
- Original repo: github.com/CATH-SWISSMODEL/cathsm-server
- We use for the workshop: git.scicore.unibas.ch/bienert/cathsm-server-sibdays2020ed.git

CATH-SM Server - Setup



CATH-SM Server - HPC Replacement



Docker: Run As Non-Root

- By default the active user inside a Docker container is root
- Generally security issue
- Also inconvenient when producing files
- Use `USER` instruction:

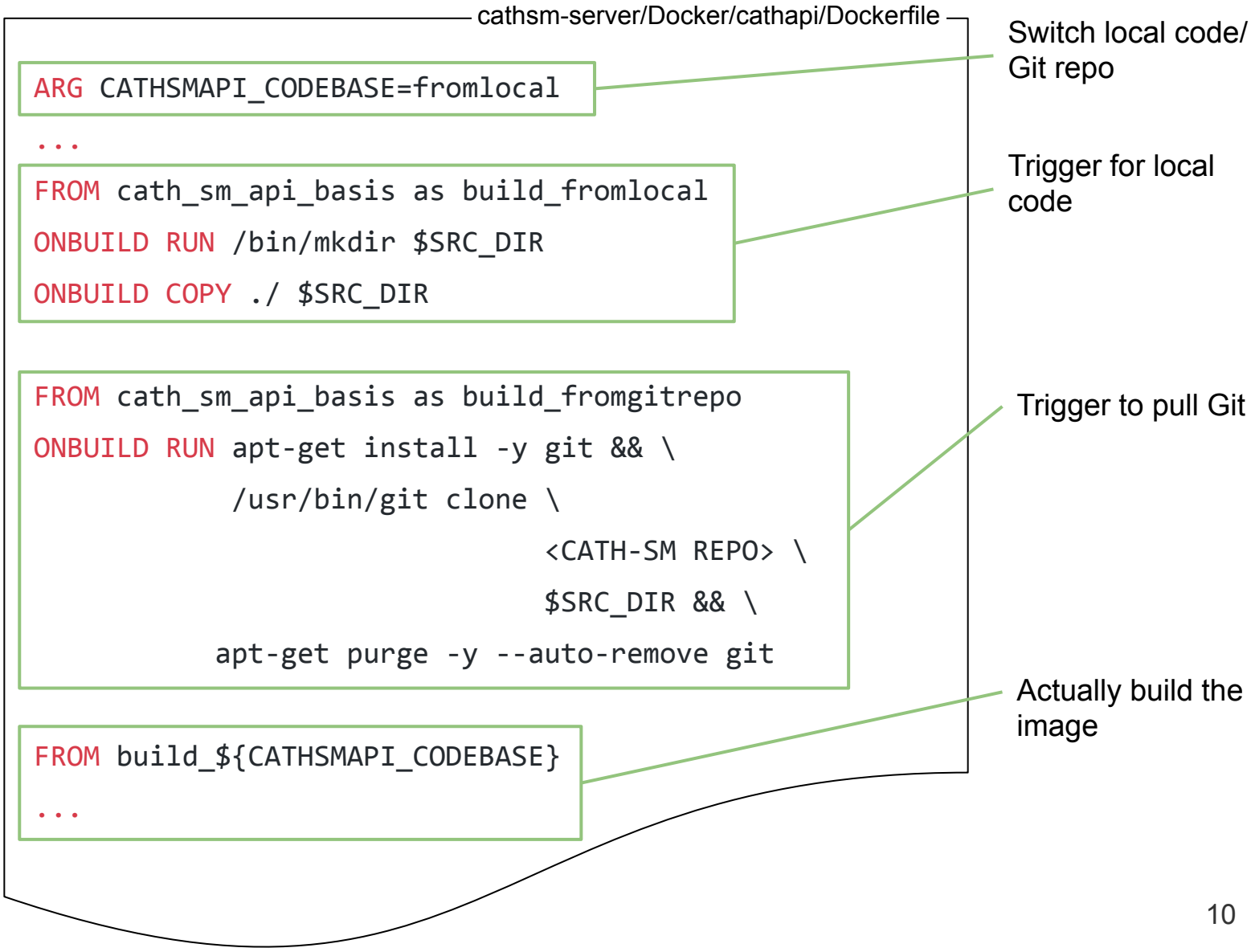
cathsm-server/Docker/nginx/Dockerfile

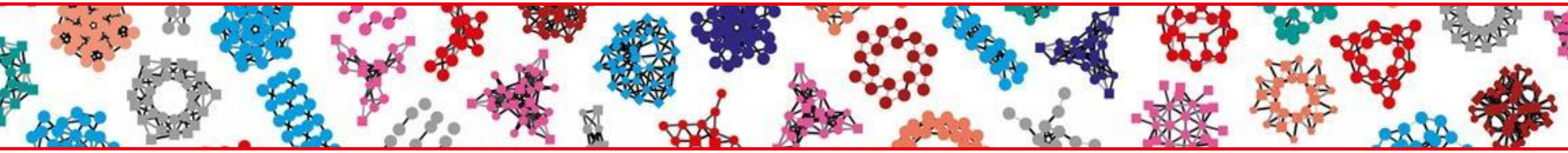
```
...
RUN adduser --system --ingroup users ucbcisi && \
    touch /var/run/nginx.pid && \
    chown -R ucbcisi:users /var/run/nginx.pid && \
    chown -R ucbcisi:users /var/cache/nginx

ADD nginx.conf /etc/nginx/nginx.conf

USER ucbcisi:users
```

Docker: Run Local Or Repo Code

- Usually Dockerfiles pull code to build from Git
- Sometimes you want to run local code in container
- Dockerfiles don't know Conditionals (`If..Then..Else`)
- But there is the `ONBUILD` instruction
- Hooks a trigger in an Docker image/ stage
- Is executed first time the image/ stage is called
- Used in the `cathapi` container of CATH-SM Server

Docker: ONBUILD





Container Orchestration

Container Orchestration

- Running single containers is simple
- Running a complete network of microservices is tough
- Orchestration automates starting a complete service, plus
  - Redundancy/ availability of containers
  - Deployment
  - Scaling
  - Resource management/ allocation
  - ...

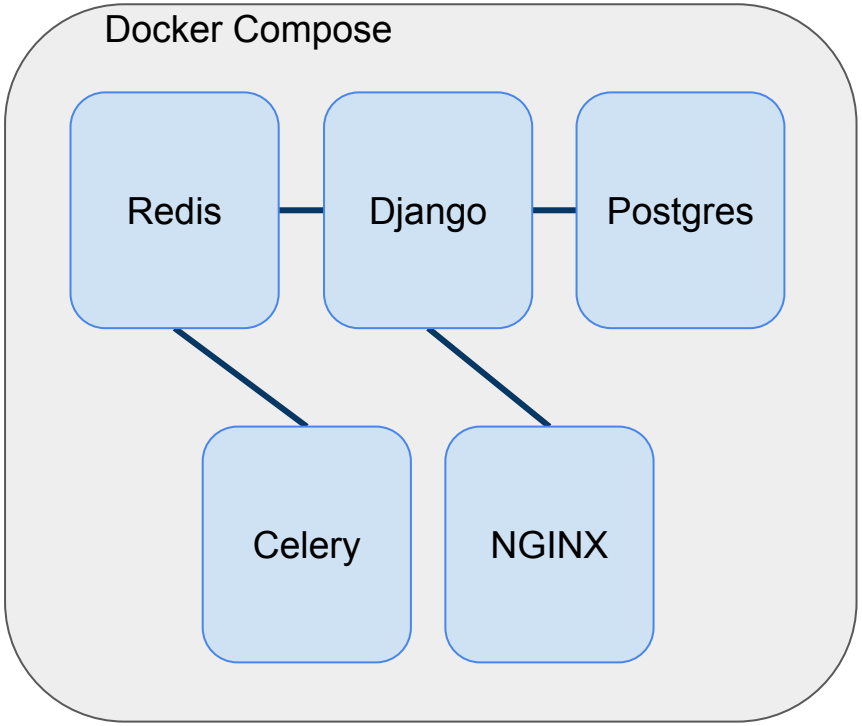
Orchestration Platforms

- Kubernetes (quasi standard)
  - Cluster/ large scale, meant for a lot of services simultaneously
  - “Feature complete”
  - Complex setup
- Docker Swarm
  - Network of Docker hosts, fitting to run multiple services simultaneously
  - “Easier than Kubernetes” (but less “powerful”)
- Docker Compose
  - Single Docker host (instance) solution, 1 service at a time
  - Easier to handle

Docker Compose

- Single instance (spreading over network would be Docker Swarm)
- Easily installed along Docker
- 1 file solution to connect multiple containers
- Common Docker CLI commands still work
- Far less features than Kubernetes but enough for a simple web application

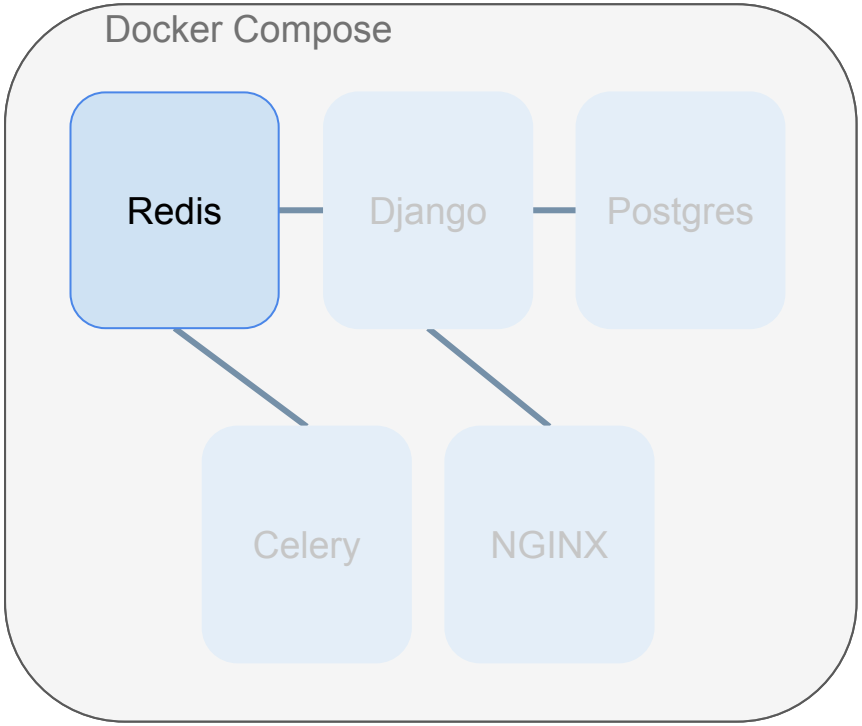
CATH-SM & Docker Compose



docker-compose.yml 1-31

```
version: "3.5"

volumes:
  db-store:
  www-static:
  www-logs:

services:
  cathapi-redis:
    restart: always
    image: $DOCKER_REGISTRY/cathapi-redis
    build:
      context: ./redis/
    networks:
      default:
        aliases:
          - cathapi-redis
```

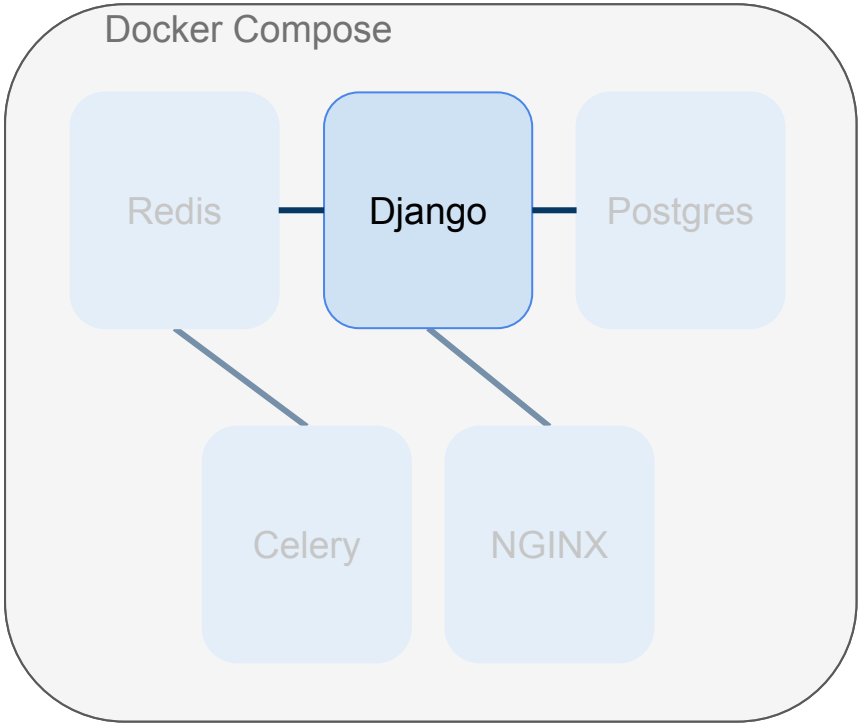
CATH-SM & Docker Compose



docker-compose.yml 20-31

```
version: "3.5"

volumes:
  db-store:
  www-static:
  www-logs:

services:
  cathapi-redis:
    restart: always
    image: $DOCKER_REGISTRY/cathapi-redis
    build:
      context: ./redis/
    networks:
      default:
        aliases:
          - cathapi-redis
```

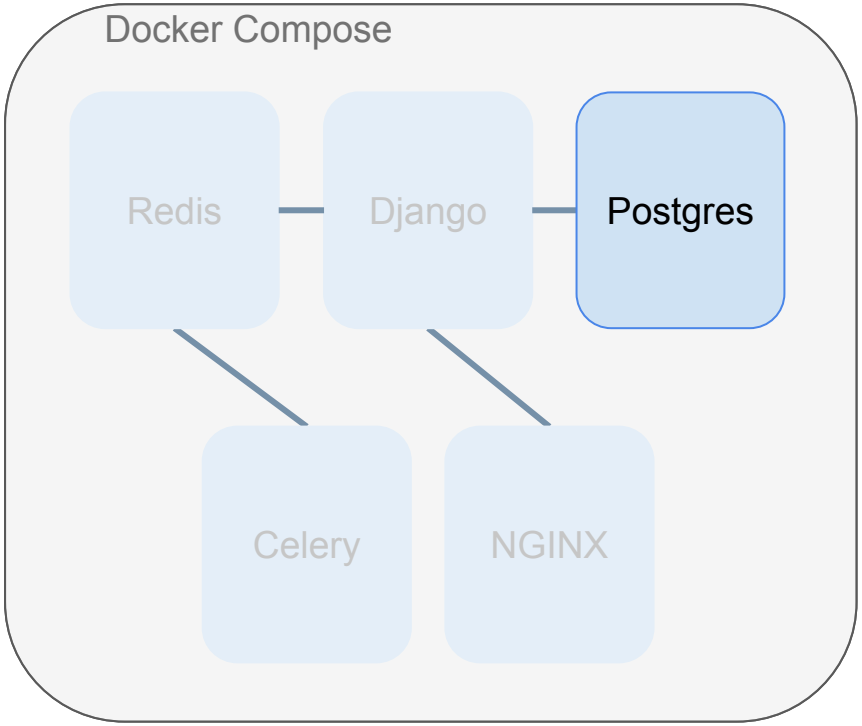
CATH-SM & Docker Compose



docker-compose.yml 32-56

```
cathapi-django:
  restart: always
  image: $DOCKER_REGISTRY/cathapi-django
  build:
    context: ../
    dockerfile: Docker/cathapi/Dockerfile
    args:
      - CATHSMAPI_CODEBASE=$CATHSMAPI_CODEBASE
      - CATHSMAPI_GITTAG=$CATHSMAPI_GITTAG
  depends_on:
    - cathapi-redis
    - postgres
  environment:
    - CATHAPI_DEBUG=CONTAINER
    - POSTGRES_DB=$POSTGRES_DB
    - ...
  volumes:
    - www-static:/static
  command: /cathapi/entrypoint.sh \
           gunicorn cathapi.wsgi -b 0.0.0.0:8080
```

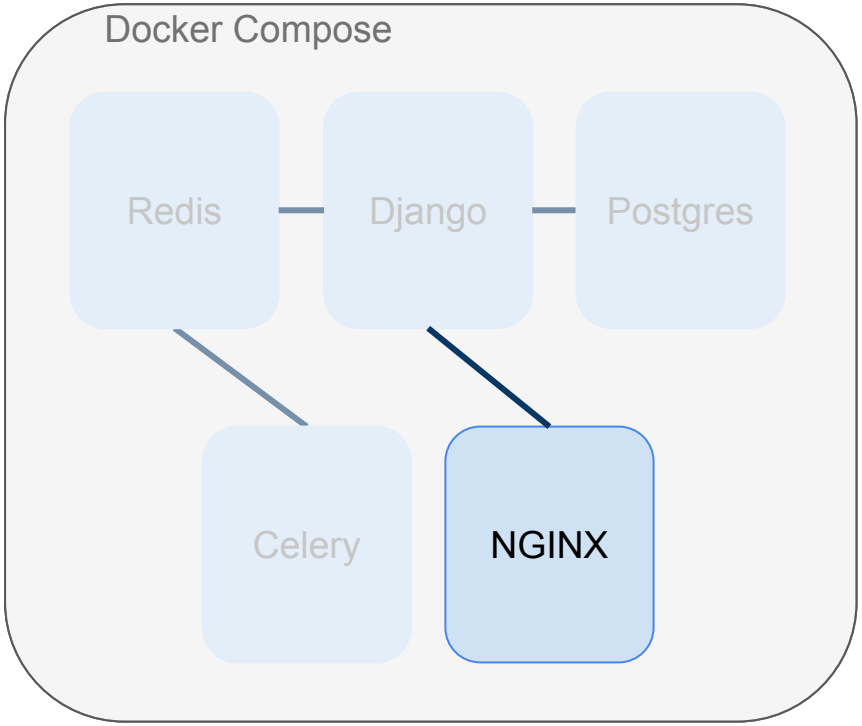
CATH-SM & Docker Compose



docker-compose.yml 57-78

```
postgres:
  restart: always
  image: $DOCKER_REGISTRY/postgres
  build:
    context: ./postgres/
    shm_size: '256MB'
  environment:
    - POSTGRES_PASSWORD=$POSTGRES_PASSWORD
    - POSTGRES_DB=$POSTGRES_DB
    - DJANGO_DB_USR=$DJANGO_DB_USR
    - DJANGO_DB_PW=$DJANGO_DB_PW
    - POSTGRES_INITDB_ARGS=--auth=scram-sha-256
  volumes:
    - db-store:/var/lib/postgresql/data/
  entrypoint:
    - docker-entrypoint.sh
    - -c
    - 'config_file=/postgresql.conf'
```

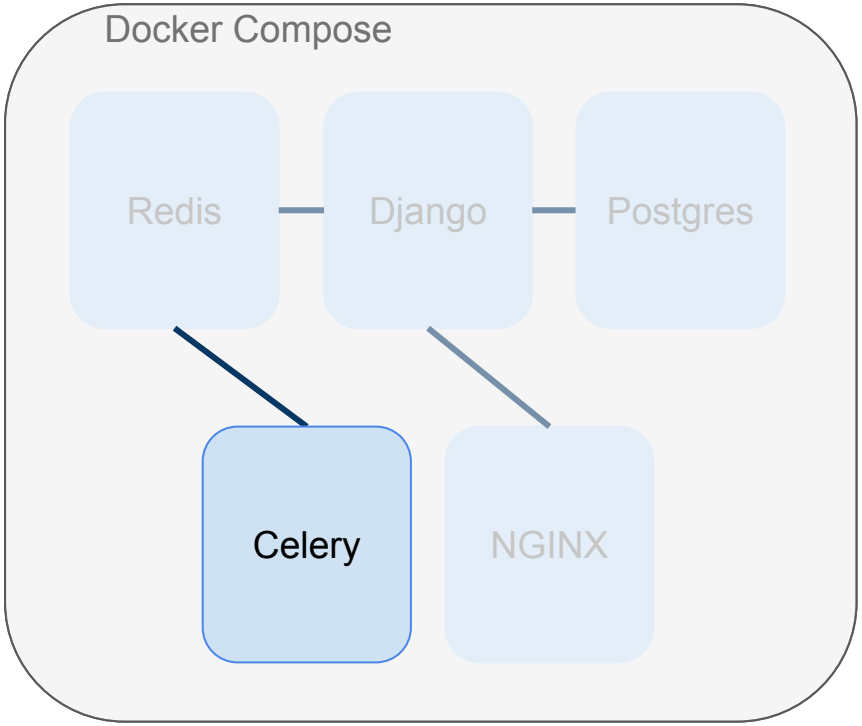
CATH-SM & Docker Compose



docker-compose.yml 79-91

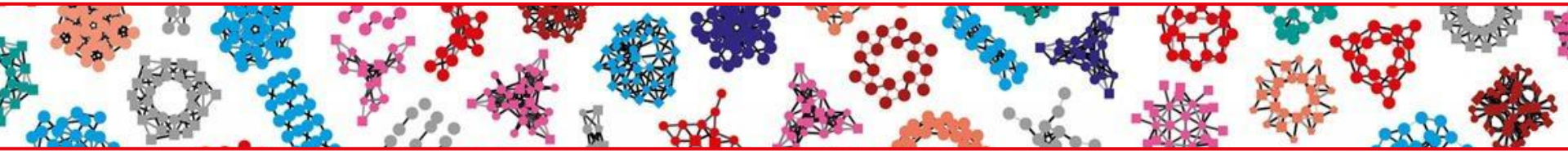
```
nginx:
  restart: always
  image: $DOCKER_REGISTRY/nginx
  build:
    context: ./nginx/
  depends_on:
    - cathapi-django
  volumes:
    - www-static:/static
    - www-logs:/var/log/nginx
  ports:
    - "0.0.0.0:80:8000"
```

CATH-SM & Docker Compose



docker-compose.yml 92-107

```
cathapi-celery:
  restart: always
  image: $DOCKER_REGISTRY/cathapi-celery
  build:
    context: ../
    dockerfile: Docker/cathapi/Dockerfile
    args:
      - CATHSMAPI_CODEBASE=$CATHSMAPI_CODEBASE
      - CATHSMAPI_GITTAG=$CATHSMAPI_GITTAG
  depends_on:
    - cathapi-redis
  environment:
    - CATHAPI_DEBUG=CONTAINER
    - I_AM_CELERY=1
  command: celery -A cathapi worker
```



CI / CD

CI / CD

- **Continuous Integration**: run an automatic pipeline with scripts for building and testing your application on every push to the repository

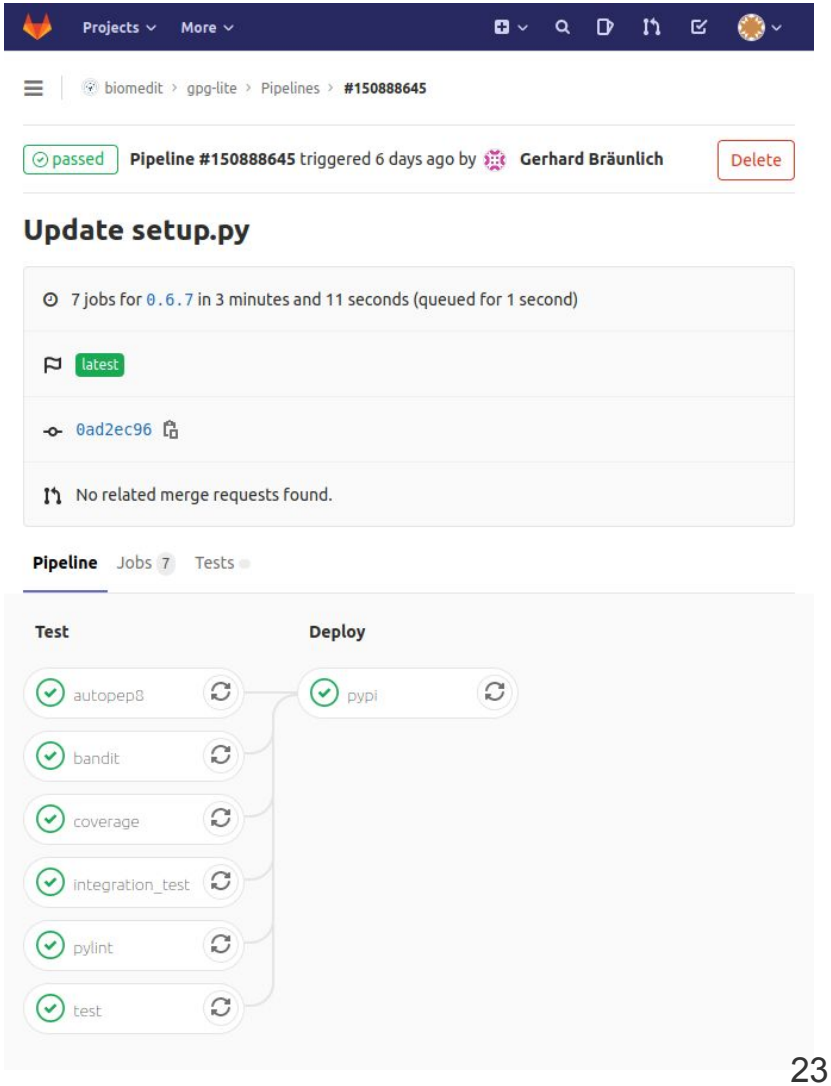
- **Continuous Deployment/Delivery**: automatically deploy your application to production on every push to the default branch (or on a new tag)

- Benefits:
  - Catch bugs early in development
  - Ensure that every commit complies with the standards defined for the project (tests, linters...)
  - No more manual deployments

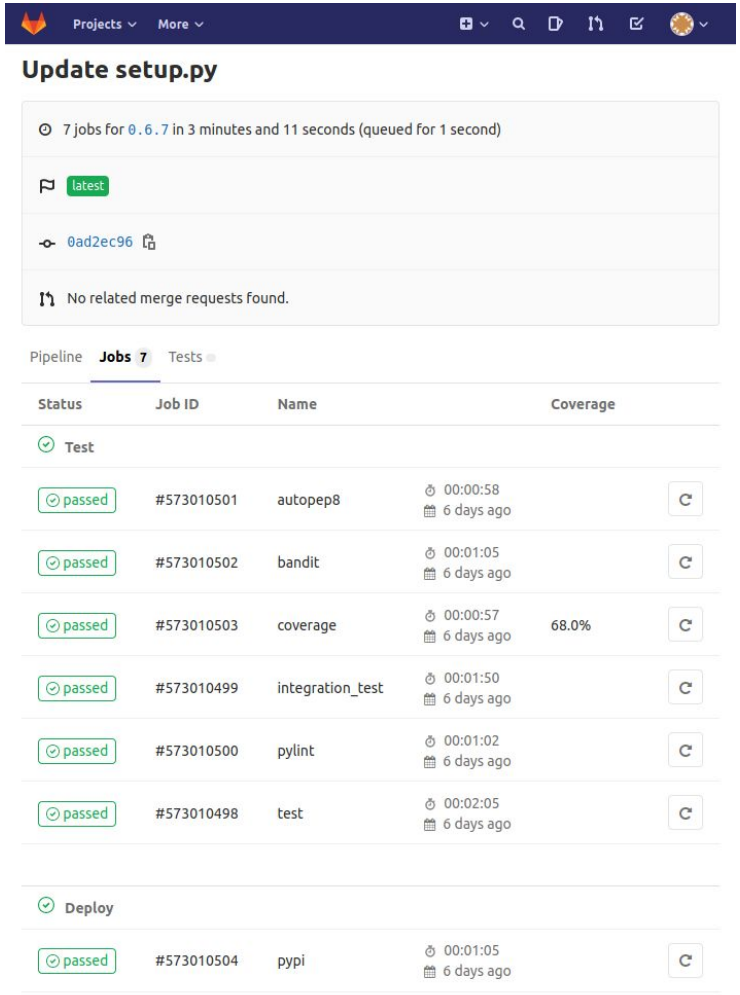
- Tools: GitLab CI, GitHub Actions,Travis CI, Jenkins...

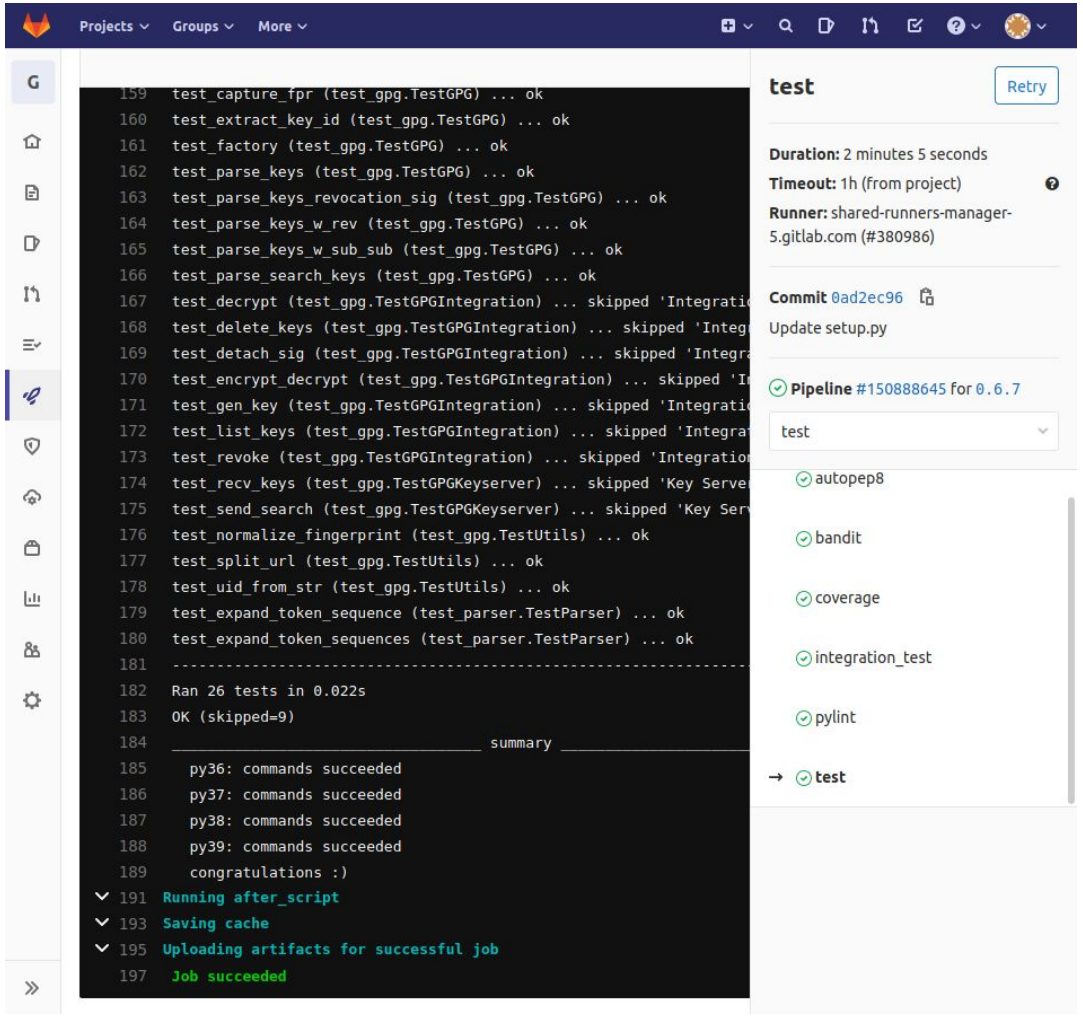
GitLab CI - Configuration

```yaml
image: python:3

pylint:
  stage: test
  before_script:
    - pip install pylint
  script:
    - pip install -e .
    - pylint --rcfile=.pylintrc --output-format=text gpg_lite/ test/

bandit:
  stage: test
  before_script:
    - pip install bandit
  script:
    - bandit -r gpg_lite/

test:
  stage: test
  image: quay.io/python-devs/ci-image:latest
  script:
    - tox

integration_test:
  stage: test
  image: dccch/gpg-lite-test-container
  script:
    - bash integration_test/test.sh
  only:
    - tags

pypi:
  stage: deploy
  variables:
    TWINE_USERNAME: $PYPI_USERNAME
    TWINE_PASSWORD: $PYPI_PASSWORD
  before_script:
    - pip install twine wheel
  script:
    - ./setup.py sdist bdist_wheel
    - twine upload --repository-url $PYPI_REPOSITORY_URL dist/*
  only:
    - tags
```



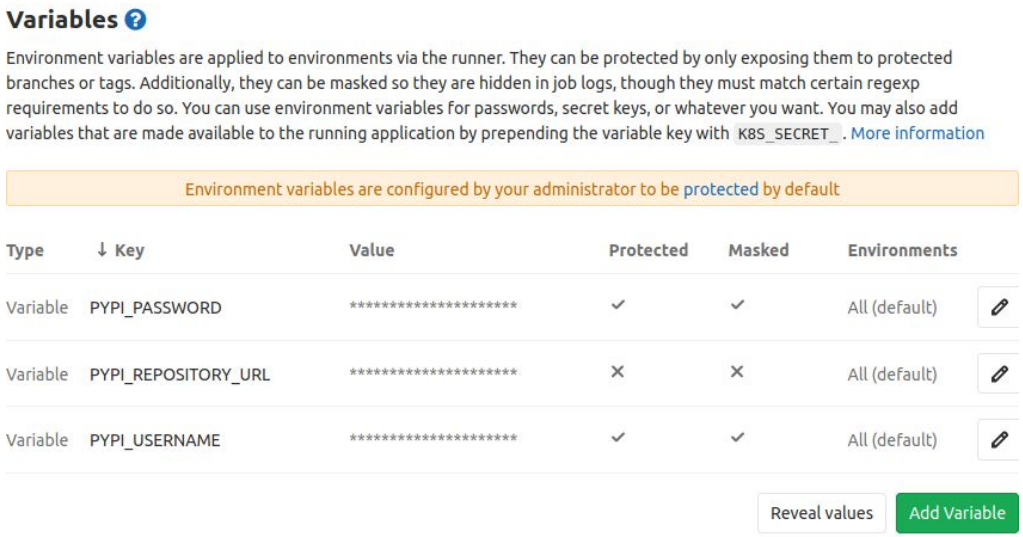
GitLab CI - Job Details Example





GitLab CI - Secrets

```yaml
image: python:3

pylint:
  stage: test
  before_script:
    - pip install pylint
  script:
    - pip install -e .
    - pylint --rcfile=.pylintrc --output-format=text gpg_lite/ test/

bandit:
  stage: test
  before_script:
    - pip install bandit
  script:
    - bandit -r gpg_lite/

test:
  stage: test
  image: quay.io/python-devs/ci-image:latest
  script:
    - tox

integration_test:
  stage: test
  image: dccch/gpg-lite-test-container
  script:
    - bash integration_test/test.sh
  only:
    - tags

pypi:
  stage: deploy
  variables:
    TWINE_USERNAME: $PYPI_USERNAME
    TWINE_PASSWORD: $PYPI_PASSWORD
  before_script:
    - pip install twine wheel
  script:
    - ./setup.py sdist bdist_wheel
    - twine upload --repository-url $PYPI_REPOSITORY_URL dist/*
  only:
    - tags
```



GitLab CI - Runners