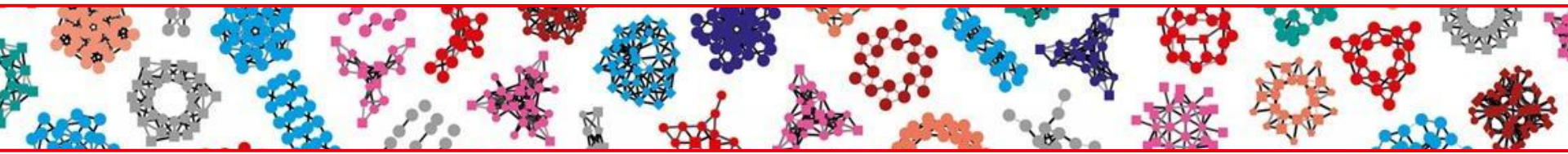


Swiss Institute of
Bioinformatics

Beyond the usual Docker tutorial

Web apps & CI

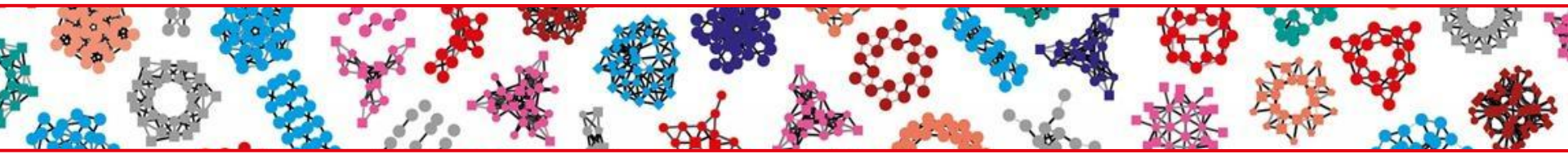
Overview



01 — **Example Application**

02 — **Container Orchestration**

03 — **CI / CD**



Example Application

Connecting web services from CATH and SWISS-MODEL
to generate 3D models from protein sequences.

▶ Try it out

The cathsm modelling pipeline can be accessed via:

Web (ReactJS) (under development)

 CATH-SWISSMODEL/cathsm-client

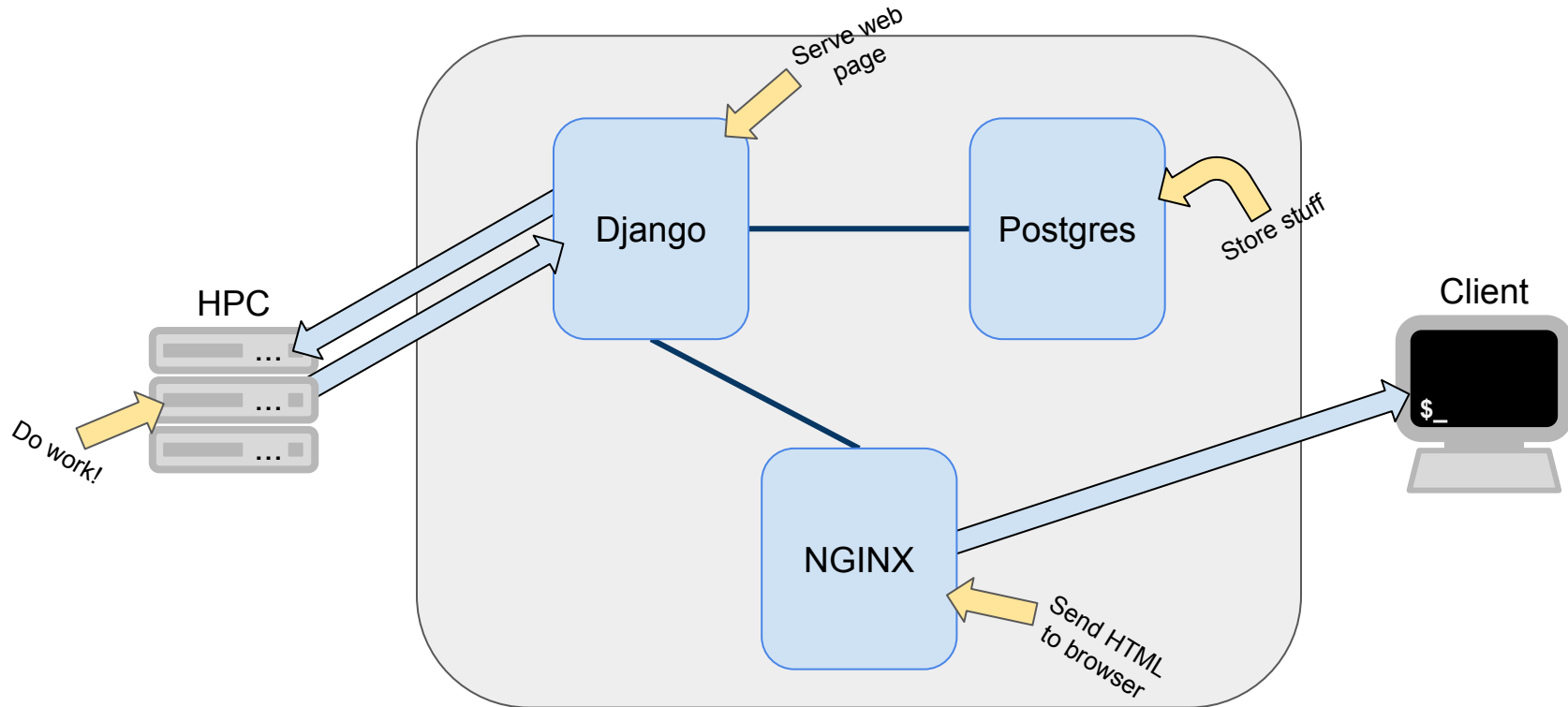
 [CATH-SWISSMODEL/cathsm-reactjs](https://github.com/CATH-SWISSMODEL/cathsm-reactjs)

ABOUT

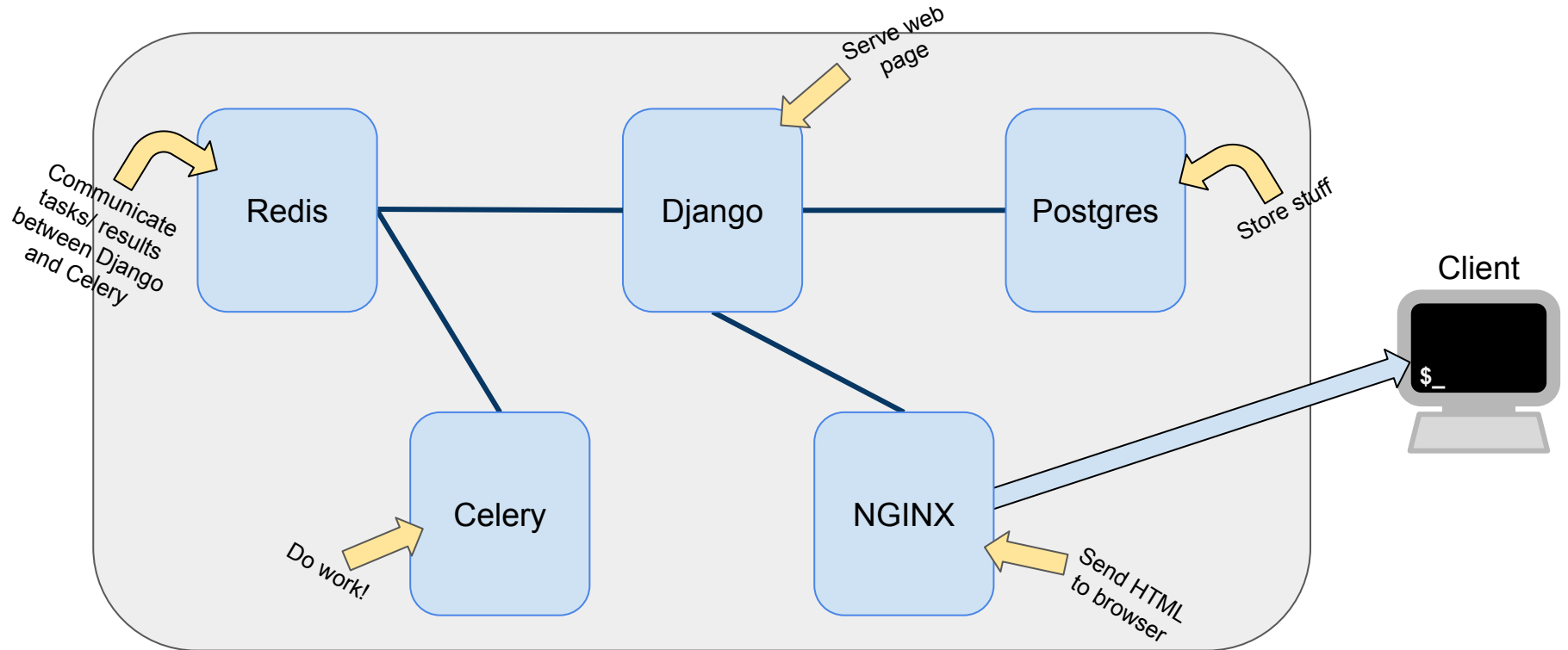
Cath-SM Server

- 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:

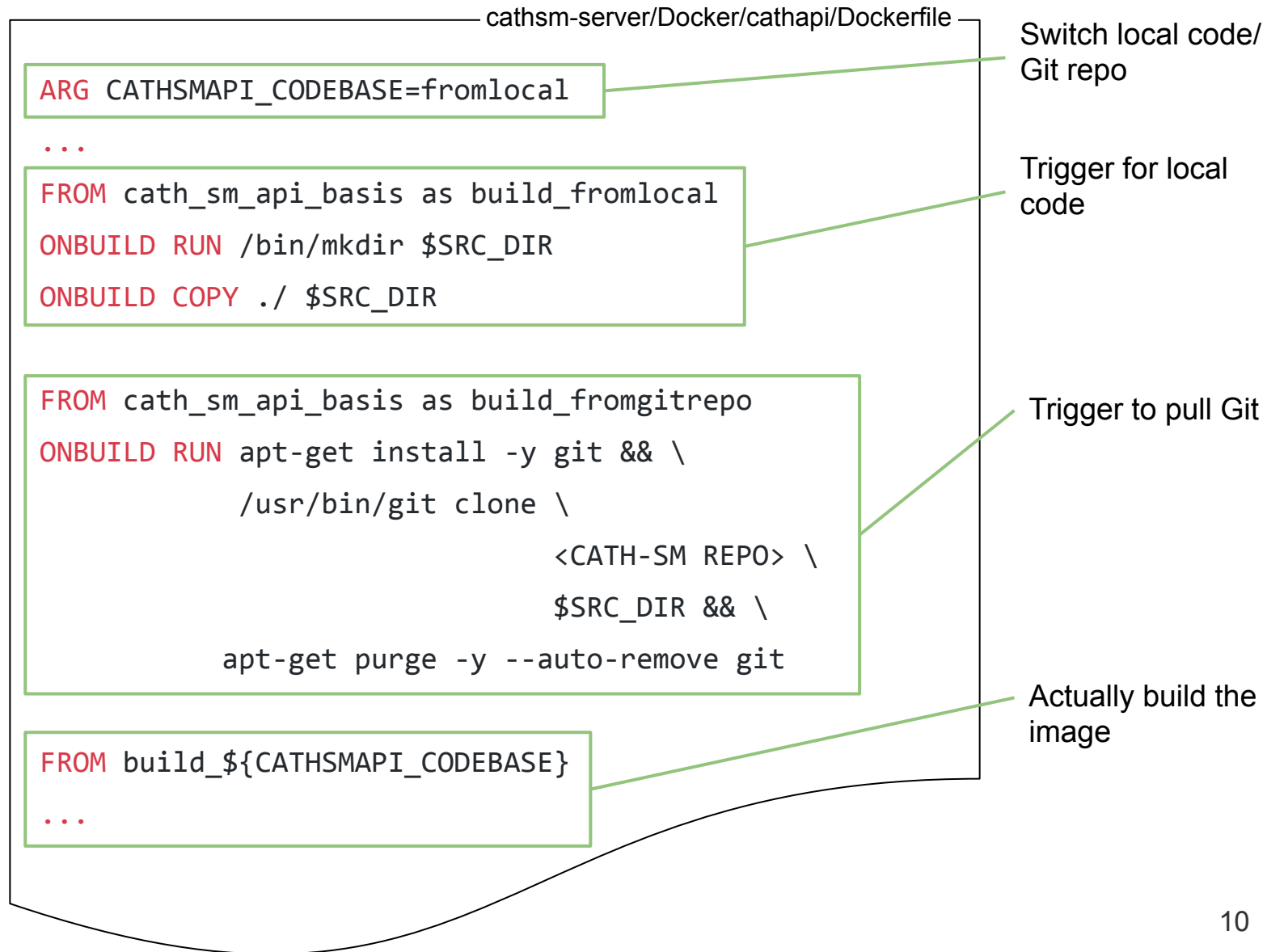
cathsm-server/Docker/nginx/Dockerfile

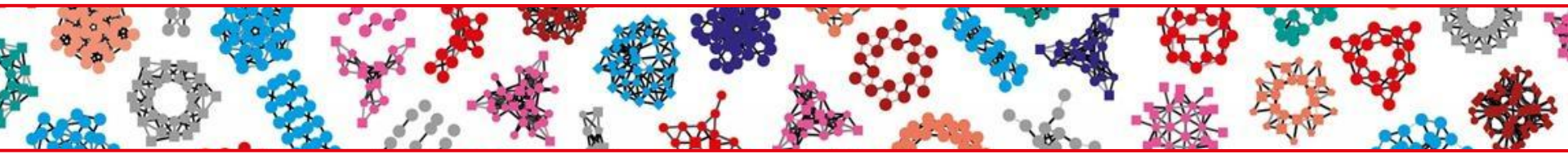
```
...  
RUN adduser --system --ingroup users ucbsisi && \  
    touch /var/run/nginx.pid && \  
    chown -R ucbsisi:users /var/run/nginx.pid && \  
    chown -R ucbsisi:users /var/cache/nginx  
  
ADD nginx.conf /etc/nginx/nginx.conf  
  
USER ucbsisi:users
```


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
 - ...

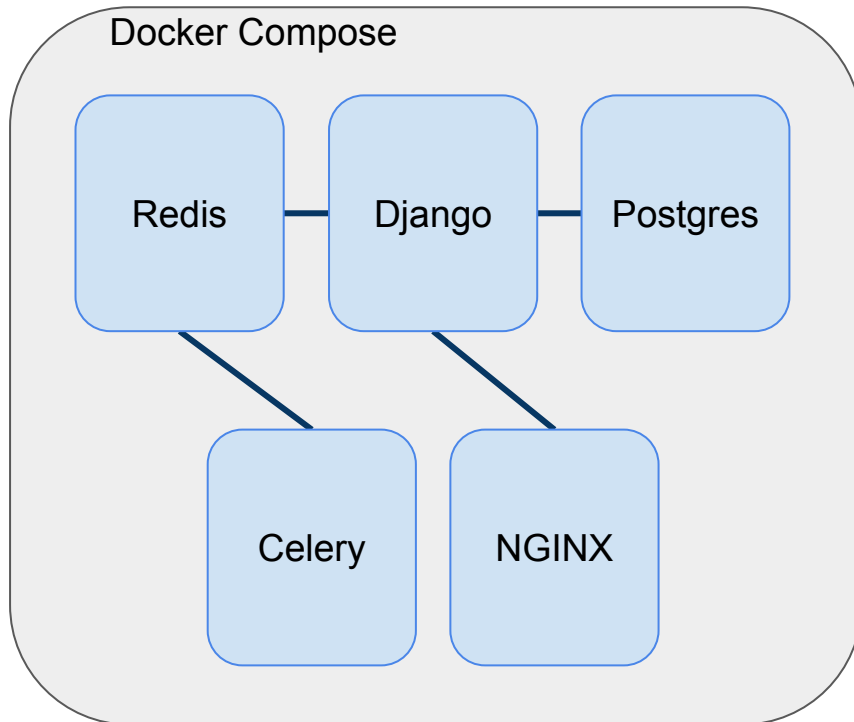
Orchestration Platforms

- 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/
```

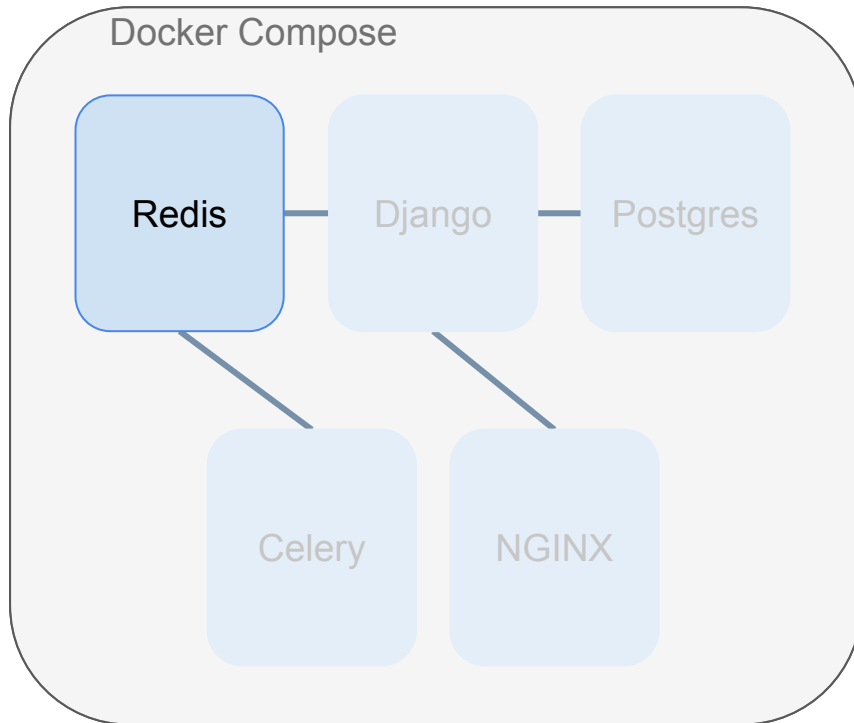
```
    networks:
```

```
      default:
```

```
        aliases:
```

```
          - cathapi-redis
```

CATH-SM & Docker Compose



version: "3.5"

volumes:

db-store:

www-static:

www-logs:

services:

cathapi-redis:

restart: always

image: \$DOCKER_REGISTRY/cathapi-redis

build:

context: ./redis/

networks:

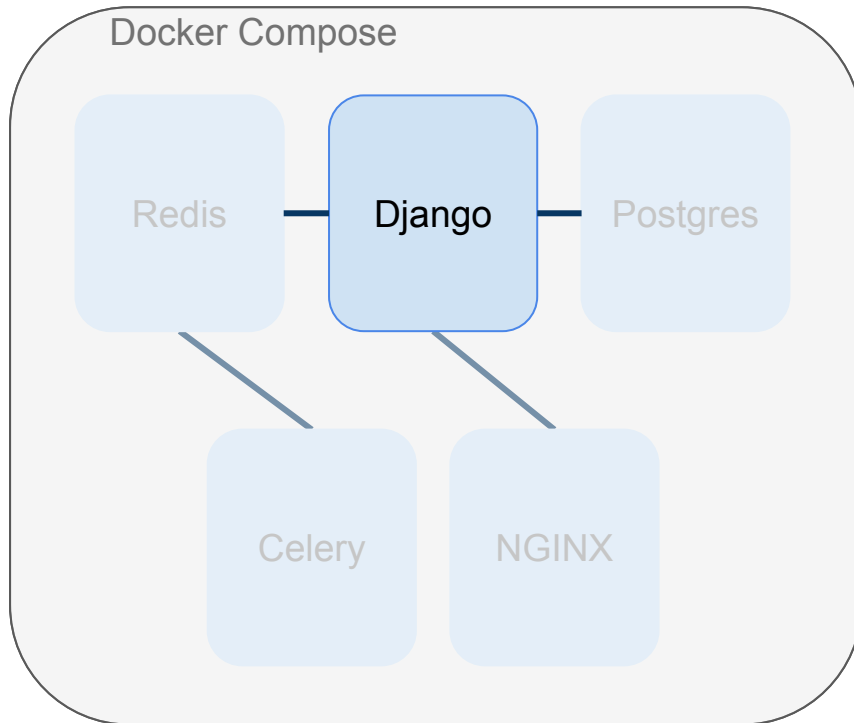
default:

aliases:

- cathapi-redis

docker-compose.yml 20-31

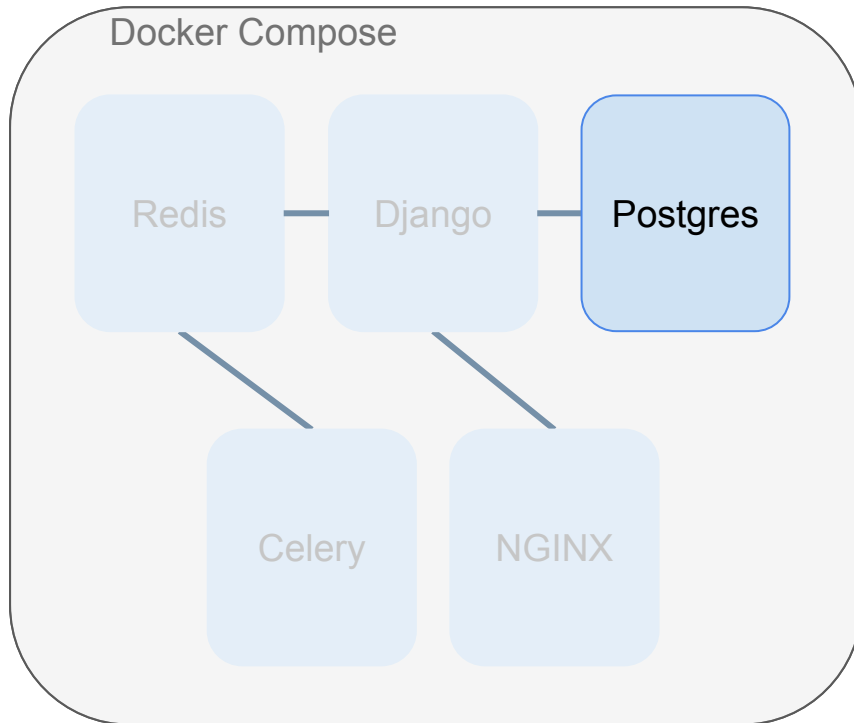
CATH-SM & Docker Compose



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
```

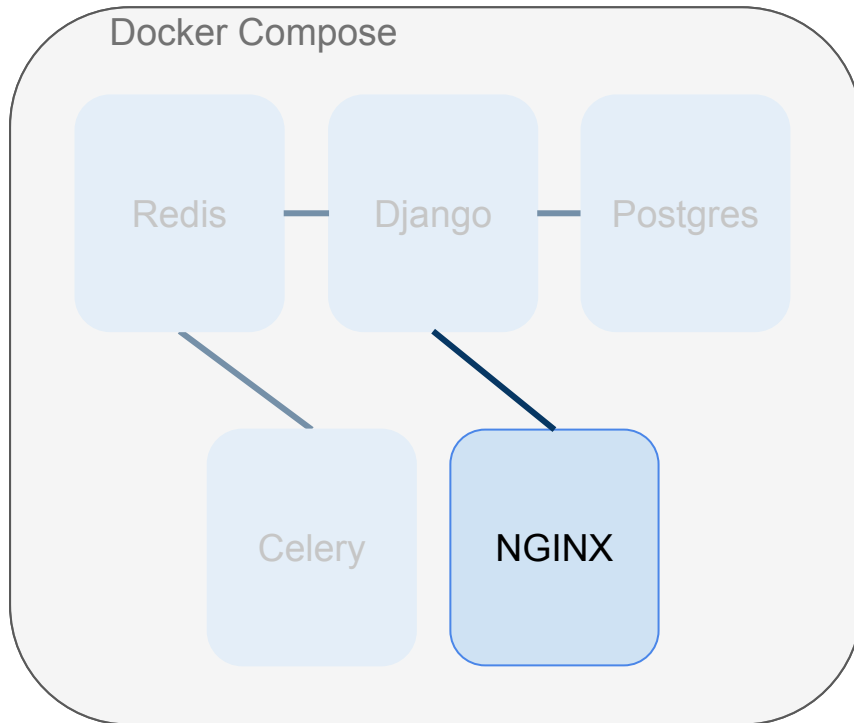
CATH-SM & Docker Compose



`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'
```

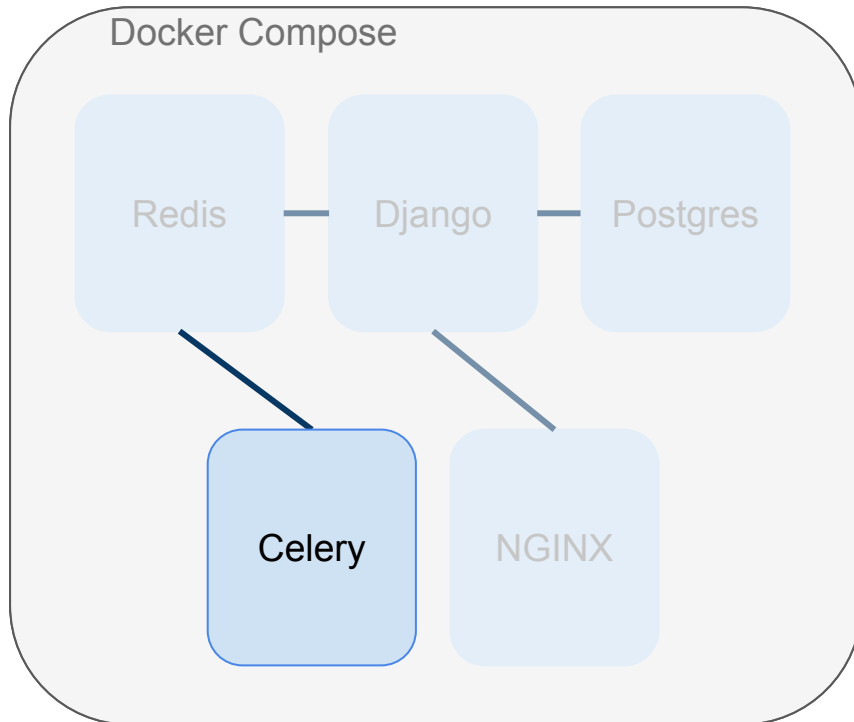
CATH-SM & Docker Compose



docker-compose.yml 79-91

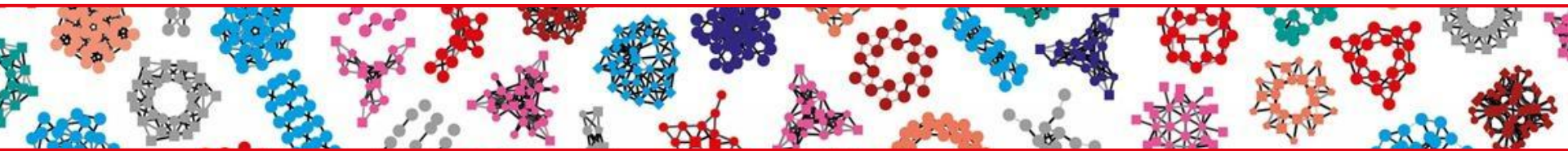
```
nginx:
  restart: always
  image: $DOCKER_REGISTRY/nginx
  build:
    context: ./nginx/
  depends_on:
    - cathapi-django
  volumes:
    - www-static:/static
    - www-logs:/var/log/nginx
  ports:
    - "0.0.0.0:80:8000"
```

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

```
1 image: python:3
2
3 pylint:
4   stage: test
5   before_script:
6     - pip install pylint
7   script:
8     - pip install -e .
9     - pylint --rcfile=.pylintrc --output-format=text gpg_lite/ test/
10
11 bandit:
12   stage: test
13   before_script:
14     - pip install bandit
15   script:
16     - bandit -r gpg_lite/
17
18 test:
19   stage: test
20   image: quay.io/python-devs/ci-image:latest
21   script:
22     - tox
23
24 integration_test:
25   stage: test
26   image: dccch/gpg-lite-test-container
27   script:
28     - bash integration_test/test.sh
29   only:
30     - tags
31
32 pypi:
33   stage: deploy
34   variables:
35     TWINE_USERNAME: $PYPI_USERNAME
36     TWINE_PASSWORD: $PYPI_PASSWORD
37   before_script:
38     - pip install twine wheel
39   script:
40     - ./setup.py sdist bdist_wheel
41     - twine upload --repository-url $PYPI_REPOSITORY_URL dist/*
42   only:
43     - tags
```

The screenshot displays the GitLab CI/CD interface for a project named 'biomedit'. The top navigation bar includes 'Projects' and 'More' dropdowns, along with icons for repository, search, pipeline, merge request, and user profile. The breadcrumb trail shows the path: 'biomedit > gpg-lite > Pipelines > #150888645'. A status bar indicates the pipeline is 'passed', triggered 6 days ago by 'Gerhard Bräunlich', with a 'Delete' button. Below this, the 'Update setup.py' job is highlighted, showing it has 7 jobs for version 0.6.7, completed in 3 minutes and 11 seconds. The job details include a commit hash '0ad2ec96' and a note that no related merge requests were found. At the bottom, the 'Pipeline' section shows a list of jobs: 'Test' (including 'autopep8', 'bandit', 'coverage', 'integration_test', 'pylint', and 'test') and 'Deploy' (including 'pypi'). Each job is represented by a green checkmark icon and a refresh button.

GitLab CI - Job Details Example

Projects ▾ More ▾

Update setup.py

7 jobs for 0.6.7 in 3 minutes and 11 seconds (queued for 1 second)

latest

0ad2ec96

No related merge requests found.

Pipeline **Jobs 7** Tests

Status	Job ID	Name	Coverage
Test			
passed	#573010501	autopep8	00:00:58 6 days ago
passed	#573010502	bandit	00:01:05 6 days ago
passed	#573010503	coverage	00:00:57 6 days ago 68.0%
passed	#573010499	integration_test	00:01:50 6 days ago
passed	#573010500	pylint	00:01:02 6 days ago
passed	#573010498	test	00:02:05 6 days ago
Deploy			
passed	#573010504	pypi	00:01:05 6 days ago

Projects ▾ Groups ▾ More ▾

test

Duration: 2 minutes 5 seconds
Timeout: 1h (from project)
Runner: shared-runners-manager-5.gitlab.com (#380986)

Commit 0ad2ec96
Update setup.py

✓ Pipeline #150888645 for 0.6.7

test

- ✓ autopep8
- ✓ bandit
- ✓ coverage
- ✓ integration_test
- ✓ pylint
- ✓ test

```
159 test_capture_fpr (test_gpg.TestGPG) ... ok
160 test_extract_key_id (test_gpg.TestGPG) ... ok
161 test_factory (test_gpg.TestGPG) ... ok
162 test_parse_keys (test_gpg.TestGPG) ... ok
163 test_parse_keys_revocation_sig (test_gpg.TestGPG) ... ok
164 test_parse_keys_w_rev (test_gpg.TestGPG) ... ok
165 test_parse_keys_w_sub_sub (test_gpg.TestGPG) ... ok
166 test_parse_search_keys (test_gpg.TestGPG) ... ok
167 test_decrypt (test_gpg.TestGPGIntegration) ... skipped 'Integrati
168 test_delete_keys (test_gpg.TestGPGIntegration) ... skipped 'Integ
169 test_detach_sig (test_gpg.TestGPGIntegration) ... skipped 'Integr
170 test_encrypt_decrypt (test_gpg.TestGPGIntegration) ... skipped 'Int
171 test_gen_key (test_gpg.TestGPGIntegration) ... skipped 'Integrati
172 test_list_keys (test_gpg.TestGPGIntegration) ... skipped 'Integrati
173 test_revoke (test_gpg.TestGPGIntegration) ... skipped 'Integrati
174 test_rcv_keys (test_gpg.TestGPGKeyserver) ... skipped 'Key Serve
175 test_send_search (test_gpg.TestGPGKeyserver) ... skipped 'Key Ser
176 test_normalize_fingerprint (test_gpg.TestUtils) ... ok
177 test_split_url (test_gpg.TestUtils) ... ok
178 test_uid_from_str (test_gpg.TestUtils) ... ok
179 test_expand_token_sequence (test_parser.TestParser) ... ok
180 test_expand_token_sequences (test_parser.TestParser) ... ok
181 -----
182 Ran 26 tests in 0.022s
183 OK (skipped=9)
184 ----- summary -----
185 py36: commands succeeded
186 py37: commands succeeded
187 py38: commands succeeded
188 py39: commands succeeded
189 congratulations :)
190
191 Running after_script
192 Saving cache
193
194 Uploading artifacts for successful job
195
196 Job succeeded
```

GitLab CI - Secrets

```
1 image: python:3
2
3 pylint:
4   stage: test
5   before_script:
6     - pip install pylint
7   script:
8     - pip install -e .
9     - pylint --rcfile=.pylintrc --output-format=text gpg_lite/ test/
10
11 bandit:
12   stage: test
13   before_script:
14     - pip install bandit
15   script:
16     - bandit -r gpg_lite/
17
18 test:
19   stage: test
20   image: quay.io/python-devs/ci-image:latest
21   script:
22     - tox
23
24 integration_test:
25   stage: test
26   image: dccch/gpg-lite-test-container
27   script:
28     - bash integration_test/test.sh
29   only:
30     - tags
31
32 pypi:
33   stage: deploy
34   variables:
35     TWINE_USERNAME: $PYPI_USERNAME
36     TWINE_PASSWORD: $PYPI_PASSWORD
37   before_script:
38     - pip install twine wheel
39   script:
40     - ./setup.py sdist bdist_wheel
41     - twine upload --repository-url $PYPI_REPOSITORY_URL dist/*
42   only:
43     - tags
```

Variables ?

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they can be masked so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. [More information](#)

Environment variables are configured by your administrator to be **protected** by default

Type	↓ Key	Value	Protected	Masked	Environments
Variable	PYPI_PASSWORD	*****	✓	✓	All (default) 
Variable	PYPI_REPOSITORY_URL	*****	✗	✗	All (default) 
Variable	PYPI_USERNAME	*****	✓	✓	All (default) 

Reveal values

Add Variable

GitLab CI - Runners

Specific Runners

Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster.

[Learn more about Kubernetes](#)

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Install Runner on Kubernetes

Set up a specific Runner manually

1. [Install GitLab Runner](#)
2. Specify the following URL during the Runner setup:
`https://git.scicore.unibas.ch/`
3. Use the following registration token during setup:



Reset runners registration token

4. Start the Runner!

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab instance does not provide any shared Runners yet. Instance administrators can register shared Runners in the admin area.

Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

This project does not belong to a group and can therefore not make use of group Runners.

Runners activated for this project

QtcyPyVp...

Pause Remove Runner

sib-days-2020-docker

#57

docker

Q56bsJFk...

Pause Remove Runner

sib-days-2020

#56

production