# Testing & CI/CD

|  |  |  |
| --- | --- | --- |
| Topic | Command | Info |
| AssertionError | Assert(a=b) | If assert true does nothing, else raises assertion error |
| Unittest |  | Automated test runner |
| Testing Django | Python manage.py test | Django TestCase creates a separate database where we can test with dummy entries. These entries do not affect the real database. Use def setUp to initialize the database. |
| Testing: Client | c = Client()  response = c.get(“/index/”)  self.assertEqual(response.statuscode, 200)  self.assertEqual(response.context[“flights”].count(), 3) | The Django testing framework also gives access to the a artificial client to test server response. We can therefore test the response code when accessing the site as well as the context information we passed to the HTML in views like data, etc. |
| Testing: Max ID | User.objects.all.aggregate(Max(“id))[“id\_\_max”] | We can get the highest primary key id using id\_\_max in django |
| Testing: Client Side using Selenium (46) | driver.get(uri)  myVar = driver.find\_element\_by\_id(“myId”)  myVar.click | Selenium can be used for client-side testing by simulating a webdriver. For that we need the uri of the html file. Elements can be accessed i.e. via id, tag name, class for example. Finally the .click method simulates a click event. All of those functions can be used for automated testing because they are run in python 🡺 use unittest combined with Selenium. |
| CI/CD | CI: Frequent merges to main branch + automated Unittesting  CD: Short release schedule | Best practices for how to write code, how it works together and how it is delivered. For continuous integration: often times many teams work on different parts of the application. Waiting to finish the branch and then merge the full code is not necessarily the best idea because if bugs happen, its hard to say where to start, therefore incremental small merge steps + unittesting at merges. |
| GitHub Actions | Use Yaml Files which are similar to JSON but also have key: list pairs as well as key:value pairs. | Github actions are run automatically when something is pushed to the code as well as styling etc. The yml. files are stored in a .github/workflows/ folders |
| Example Yml Action | name: Testing  on: push  jobs:  job-name:  runs-on: ubuntu-latest  steps:  - uses: actions/checkout@v2  - name: Run Django unit tests  run: |  pip3 install –user Django  python3 manage.py test | Name: name of the workflow 🡪 here Testing as we run the unittests  On: When should this workflow run? Here on push  Runs-on: virtual machine the job runs on, here linux  Uses: actions/checkout@v2 🡪 checks the code from the git repository out so that we can use it |
|  |  |  |
| Docker | FROM python:3  COPY . /usr/src/app  WORKDIR /usr/src/app  RUN pip install -r requirements.txt  CMD [“python3”, “manage.py”, “runserver”, “0.0.0.0:8000”] | Docker eliminates the deployment problems which happen when you move you deploy your software which works on your machine onto a different machine using virtualization software. Example code deployment file. Explanation: We build from the python image up, copy our filed to usr/src/app, cd in that folder, install all dependencies and run the server on port 8000 |
| Docker SQL | version: ‘3’  services:  db:  image: postgres    web:  build: .  volumes:  - .:/usr/src/app  ports:   * “8000:8000” | We replace the sqlite DB with postgres DB because it can handle multiple requests. For that we use docker-compose function. Here we have 2 services 1) database db using dockers postgres image and 2) our web service which is build using . 🡪 the docker file in this folder, volumes means my current directory should correspond to the app directory, ports maps port 8000 on the container to port 8000 on my machine so that I can access it via my web browser |
| Docker Login | docker ps  docker exec -it container-id bash -l | Docker ps -> shows all currently running docker containers  Docker exec runs a command in the chosen container. -it makes it interactive and bash -l creates a docker shell |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |