Deployment and DevOps

Kianoosh Abbasi

CSC309 Winter 2022

So far

- We've learned how to develop a website; backend and frontend.
- But it was always running on the local computer!
- Last session: it's time to deploy!

This session

- Deployment of a web application
  Choosing a webserver
- Django deployment
  Socket and gunicorn
- React deployment
  Build and serve
- Course conclusion: DevOps

What is a server?

- It's another computer (or virtual machine), like your local one!

- It should have a static IP address
  - And maybe a domain bound to that IP

- But that's not about it!

Development vs Production

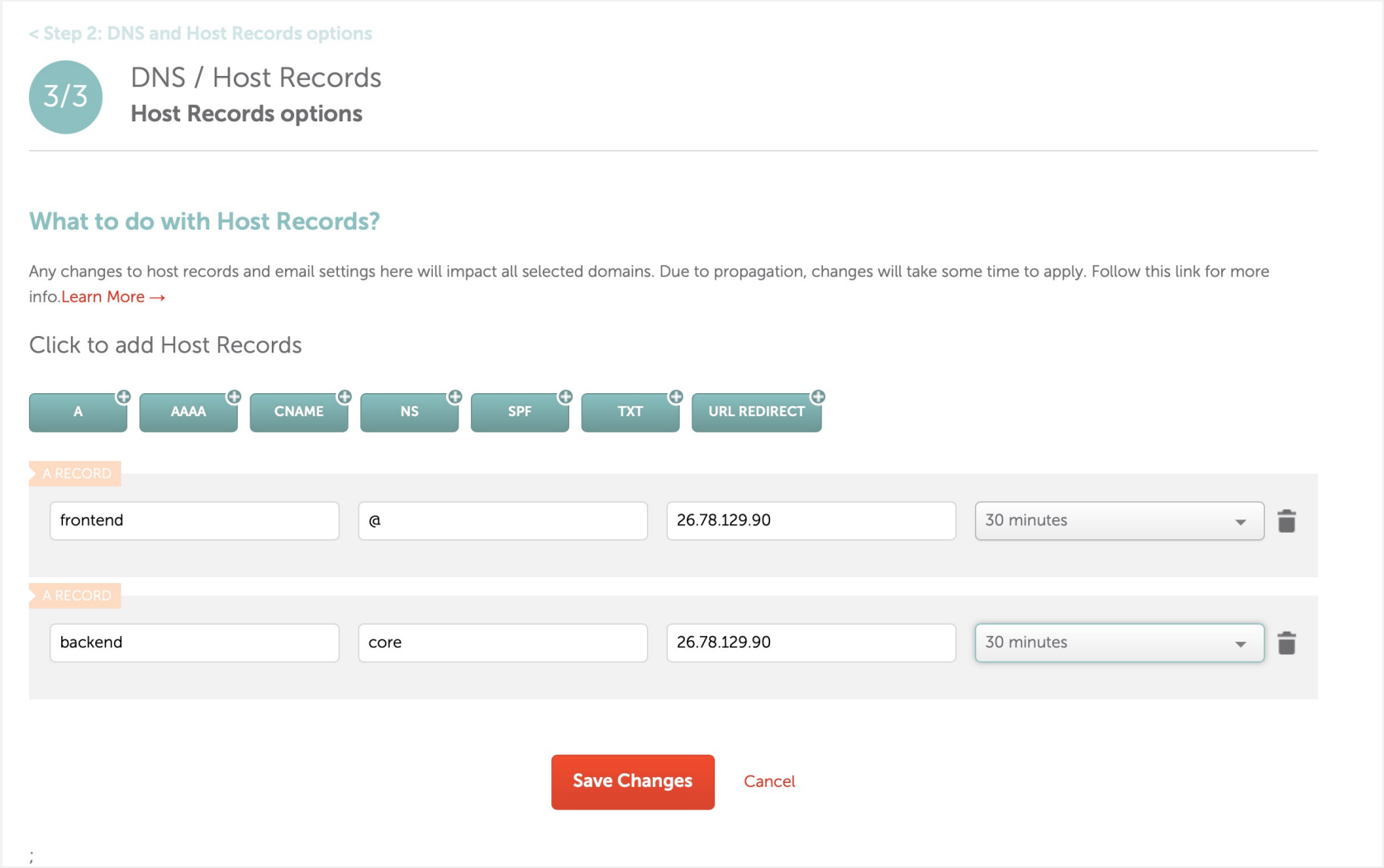
- IP only determines how your application can be accessed

- You can access your local website within the same network

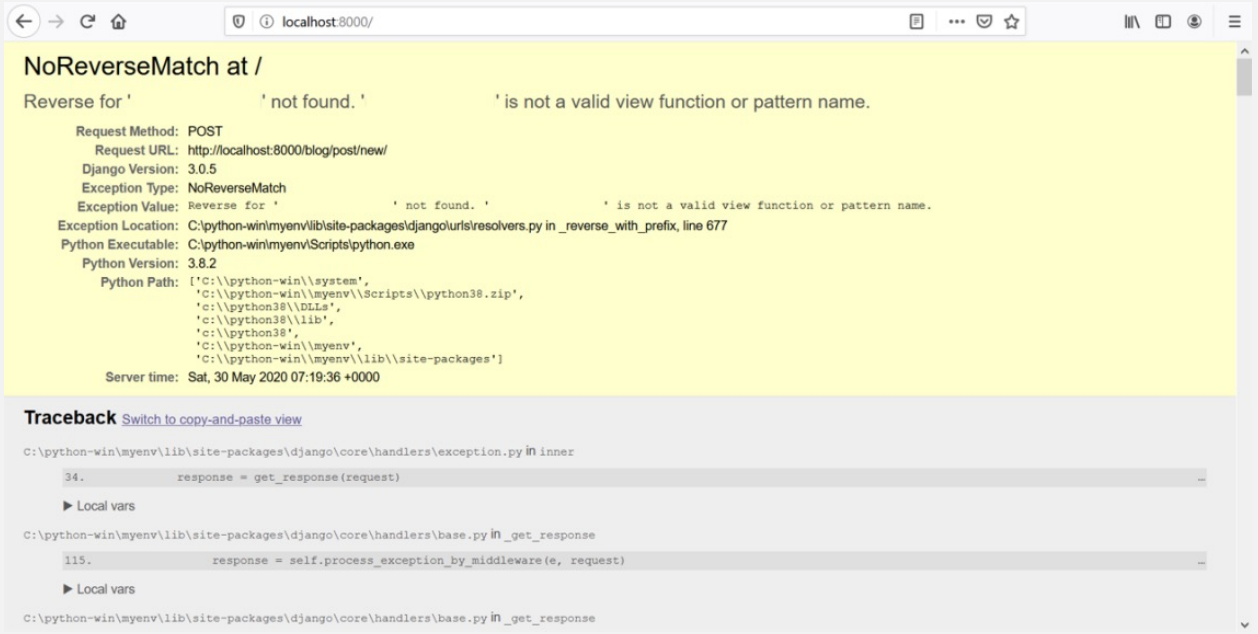
  Just need to run `./manage.py runserver 0.0.0.0`

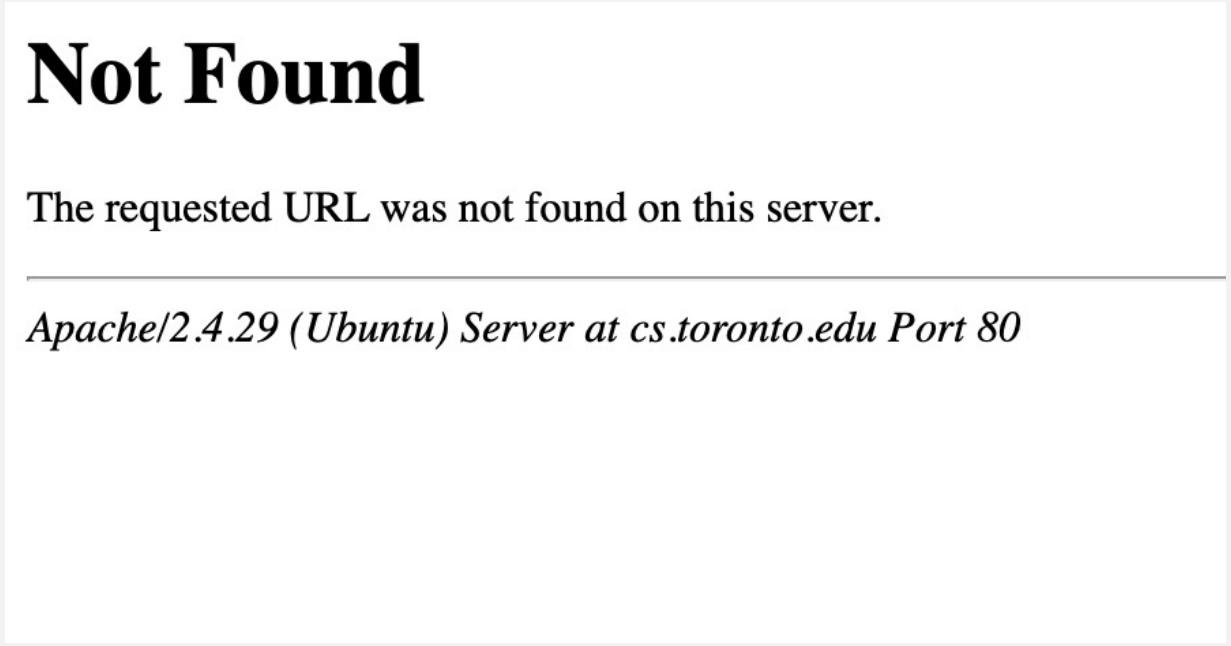
- Development vs production is about how you run your application!

Get a domain


< Step 2: DNS and Host Records options
3/3
DNS / Host Records
Host Records options
What to do with Host Records?
Any changes to host records and email settings here will impact all selected domains. Due to propagation, changes will take some time to apply. Follow this link for more info.Learn More →
Click to add Host Records
A
AAAA
CNAME
NS
SPF
TXT
URL REDIRECT
A RECORD
frontend
@
26.78.129.90
30 minutes
A RECORD
backend
core
26.78.129.90
30 minutes
Save Changes
Cancel
;

Development vs Production





https://www.dark-hamster.com/application/how-to-solve-error-message-noreversematch-at-url-in-django/

Development vs Production

- Uses a light-weight database (SQLite)
- Upon errors, the actual stack trace is shown
- Run with a development server

- Uses a real database: Postgres, MySQL, ...
- Just a generic 500 or 404 is shown
- Run with a real webserver

Deploying a Django project

General notes

- You can clone your code on your server and run manage.py
- Your website can be accessed via the server's IP address
- But it's still the development mode

Production settings

- Some of developments settings must be overridden!

- Examples

  `DEBUG`
  `SECRET_KEY`
  `ALLOWED_HOSTS`
  `DATABASE`

Production settings

- Django loads settings according to an environment variable named `DJANGO_SETTINGS_MODULE`
- Create another settings file that imports everything from `settings.py`, but overrides some
- Or create a `.env` file and load all settings from that file, for both development and production
- Note: The new settings or env file must not be on git!

Production settings

- Local and production .env files include respective information

- For development: should use a package that loads the .env file on startup

  Example: python-dotenv

```
DATABASES = {
    'default': {
        'ENGINE': os.environ['DB_ENGINE'],
        'NAME': os.environ['DB_NAME'],
        'USER': os.environ['DB_USER'],
        'PASSWORD': os.environ['DB_PASSWORD'],
        'HOST': os.environ['DB_HOST'],
        'PORT': os.environ['DB_PORT'],
    }
}
```

Static files

- Static files are file/directory access granted to the webserver

- Django collects all of them from app folders and gathers them under `STATIC_ROOT`
  - For security and performance reasons
  - They are often served by the webserver itself!

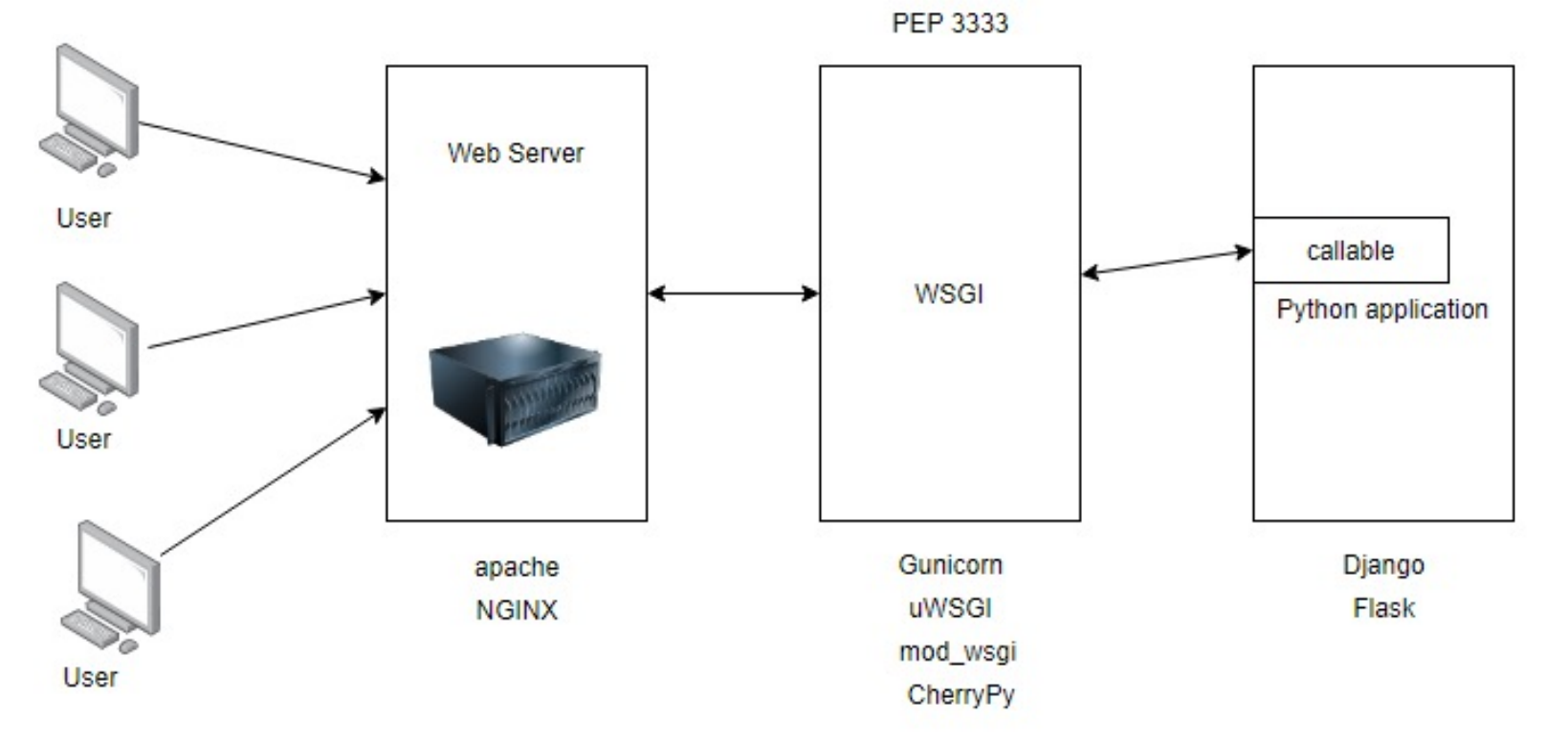
- Via command: `./manage.py collectstatic`

Webserver

- Time to say goodbye to Django's development server
- It's not meant to handle high loads, withstand attacks, etc.
- Python's solution: Web Server Gateway Interface (WSGI)

WSGI

- An interface through which requests are forwarded to your application



Source: https://medium.com/analytics-vidhya/what-is-wsgi-web-server-gateway-interface-ed2d290449e

Gunicorn

- An example WSGI server
  Installed via pip

- Run it with this command
  ```
  gunicorn --bind 0.0.0.0:8000 myproject.wsgi
  ```

- You can specify the number of workers, log level, etc.

Gunicorn

- Current execution is still not good
- Stops when you close your terminal
- Solution: make it a service!
  - Runs forever
  - Restarts upon error
  - Runs on startup

Service

- Linux has a tool to register and monitor your services!
- Create a new file under the services directory

  /etc/systemd/system/<service_name>.service

```
[Unit]
Description=first_projet gunicorn daemon
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/first_project
ExecStart=/home/ubuntu/first_project/venv/bin/gunicorn \
                --access-logfile - \
                --workers 3 \
                --bind unix:/home/ubuntu/first_project/first_project.sock \
                first_project.wsgi:application

[Install]
WantedBy=multi-user.target
```

- Note that it binds to a socket, rather than 0.0.0.0:8000
- Reason: we want a real webserver to serve our application (on port 80)
- we cannot let this gunicorn take port 80. WHY?
- A real webserver listens on port 80 and forwards certain requests to gunicorn

  Gunicorn serves the backend project through a local socket

Service

- Manage your service via these commands:

```
sudo service <name> restart
sudo service <name> status
sudo service <name> stop
```

- When you change the service file, you might need a reload

```
sudo systemctl daemon-reload
```

Webserver

- It’s time to install a webserver
  Examples: Nginx, Apache
- Listens on ports 80, 443, etc.
- Serves multiple web applications on the same computer
  Backend, frontend, other applications, etc.
- Has many security features

Deploy with Nginx

- Create your config file at /etc/nginx/sites-available/<project_name>
- Make a symbolic link to that file at /etc/nginx/sites-enabled/
- Restart Nginx!

/etc/nginx/sites-available/myproject

```
server {
    listen 80;
    server_name server_domain_or_IP;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/sammy/myproject;
    }

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/sammy/myproject/myproject.sock;
    }
}
```

That's it!

- Now, you can access your website through its domain or IP
- Debugging in the production mode can be hard
- You need to manually go through logs
- Check everything is fine after each step

Notes

- Can leave debug to true for a short time
  Don't forget to turn it back off

- Reload notes:
  When your Django code changes: restart the gunicorn service
  When the service file changes: reload and restart the service
  When nginx config changes: restart nginx

- Great, step-by-step tutorial:
  https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-16-04

Deploying a React project

General notes

- Good news: it's much easier than Django!
- Only need to build the project and configure Nginx to serve the appropriate files
- Build your React project via: `npm run build`
- The build folder appears in your project

- Nginx config: just route all requests to the build folder

- The config file (sites-available and sites-enabled):

```
server {
    listen 80;
    server_name mysite.com www.mysite.com

    root /home/ubuntu/app-deploy/build;
    location / {
        try_files $uri /index.html;
    }
}
```

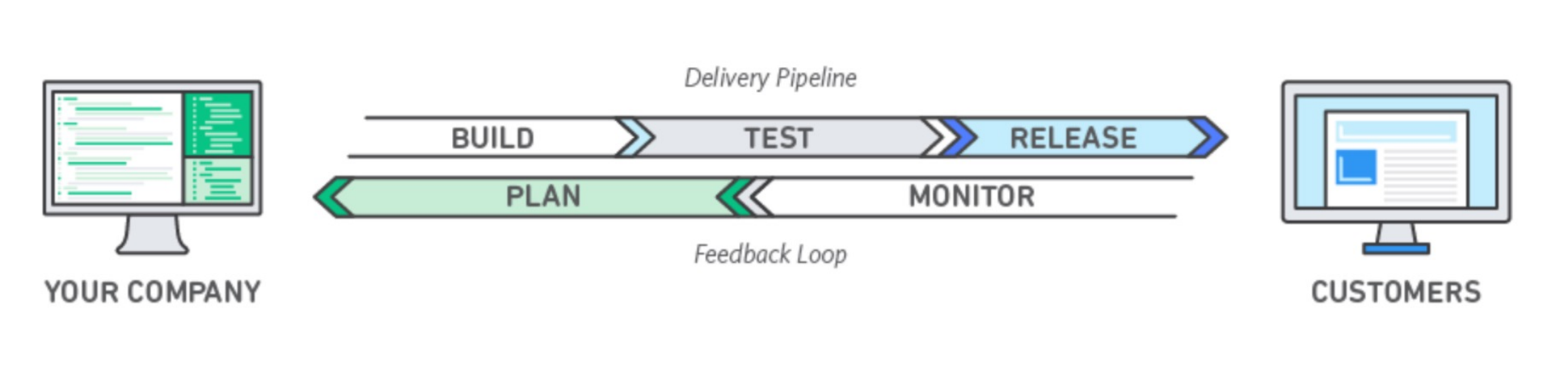
- Restart Nginx!

  Tutorial link: https://enrico-portolan.medium.com/how-to-host-react-app-with-nginx-634ad0c8425a

Questions?

DevOps

Visit https://aws.amazon.com/devops/what-is-devops



DevOps

- A set of principles and practices

- Continuous delivery:

  Automated build and deployment

- Current way of deployment is complicated and hard

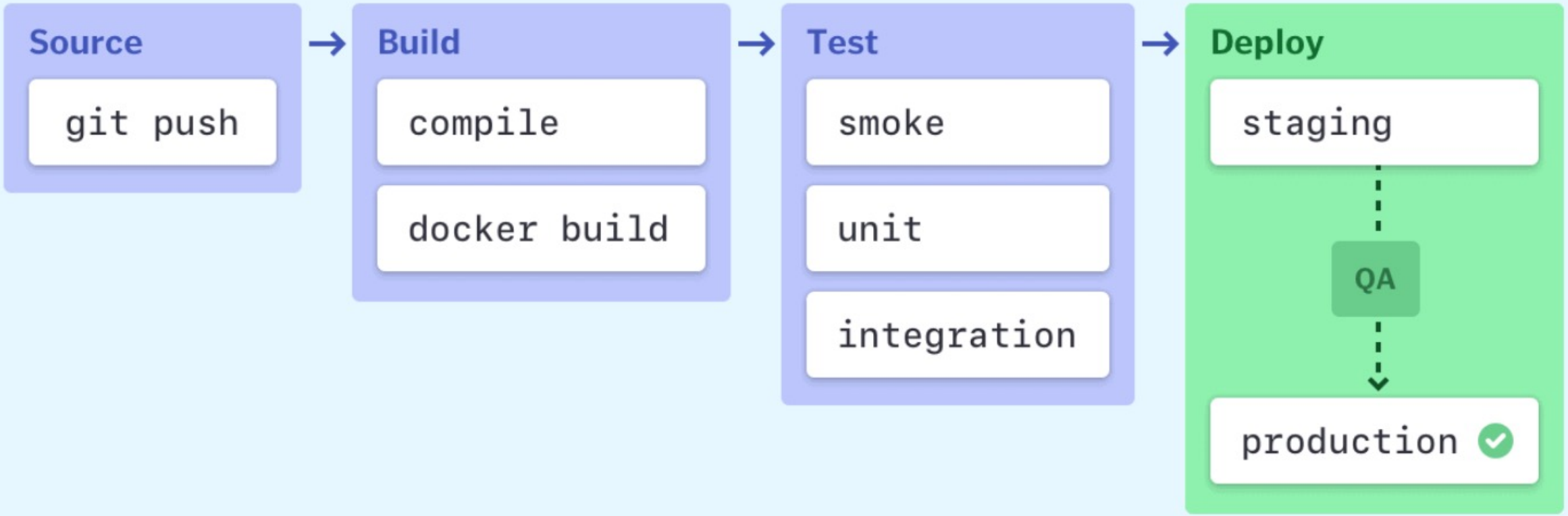
  You can move all commands to a shell script, but it's still not automated

Auto DevOps

- Github and Gitlab support auto DevOps

- Once you push, a pipeline starts
  - Test
  - Build
  - Deploy

- Also called CI/CD (continuous integration & continuous delivery)

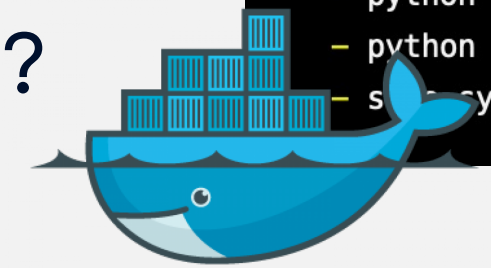


Source: https://semaphoreci.com/blog/cicd-pipeline

Example

- Create .gitlab-ci.yml
- Put all stages and scripts there!
- But who runs these??

```
deploy-staging:
  only:
    - develop
  when: manual
  script:
    - rsync -r ./ $TARGET_DIR
    - source $PY_ENV_PATH
    - cd $TARGET_DIR
    - pip install --upgrade -r requirements/$PIP_FILE.txt
    - python src/manage.py compilemessages
    - python src/manage.py collectstatic --noinput
    - python src/manage.py migrate --noinput
    - python create_env_file.py -o
    - s  systemctl restart uwsgi
```



docker

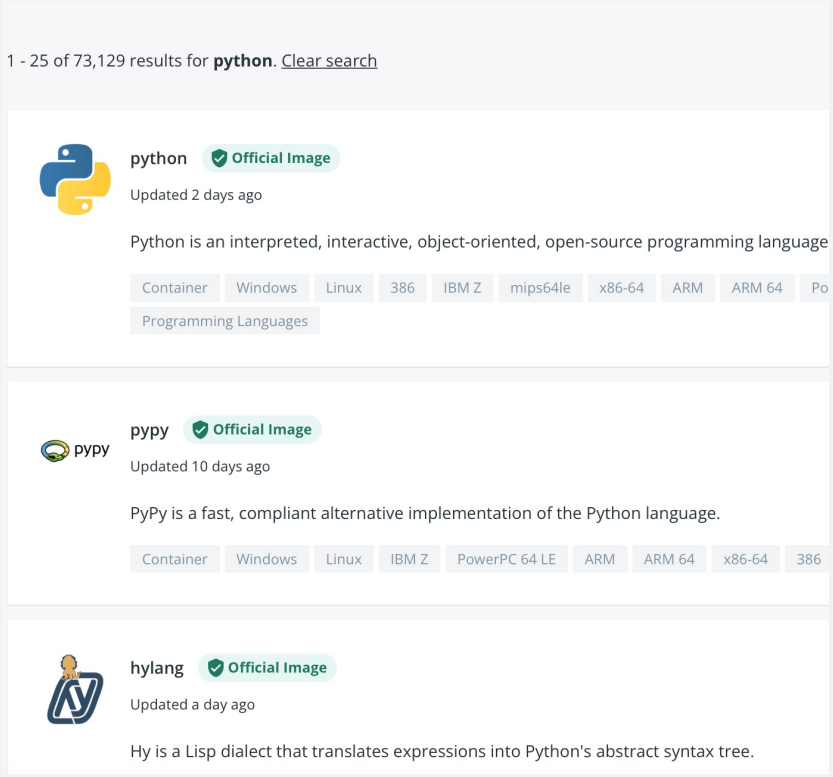
Docker

- Revolutionized software delivery

- Wraps your application in a virtualization called container
  - Somewhat like a VM, but not the same!
  - So easy to set up: download the image from the registry or build is from a Dockerfile
  - Makes your app portable: can be stopped, restarted, copied easily!

Docker

Dockerhub



Example Dockerfile

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
```

DevOps with Docker

- Github/Gitlab can run all CI/CD processes through a docker container
- If everything is fine, the docker image will be your application!
- Can be pushed to a registry and pulled in the server

Example Dockerfile for Django

Visit https://www.section.io/engineering-education/django-docker/
Also visit https://www.digitalocean.com/community/tutorials/how-to-build-a-django-and-gunicorn-application-with-docker

```
# pull the official base image
FROM python:3.8.3-alpine

# set work directory
WORKDIR /usr/src/app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN pip install --upgrade pip
COPY ./requirements.txt /usr/src/app
RUN pip install -r requirements.txt

# copy project
COPY . /usr/src/app

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

This course…

- A journey through the world of world wide web!
  - It was a long one, wasn't it?
- It's evolving at the speed of light!
  - Many of these things can go obsolete in 5 years!!
  - New technologies and innovations arrive!
- Hope you've had a fun time :)

