# Python radionica #4

## Infrastruktura i administracija servisa

10 April 2017

Janoš Guljaš
Seven Bridges Genomics

## Infrastruktura i administracija servisa

Dizajniranje, razvoj i rad servisa uključuju interakciju sa različitim ljudima i računarskim sistemima.

Često se pri razvoju zanemaruju neke od interakcija što dovodi do problema koji koče dalji razvoj ili otežavaju rad.

Podteme:

- Korisnici servisa
- Zavisnosti servisa
- Pakovanje i distribucija
- Automatizacija testiranja

## Korisnici servisa/proizvoda

Svi ljudi koji interaguju sa proizvodom na bilo koji način.

- Krajnji korisnici proizvoda
- QA inženjeri
- Mainteiner-i paketa
- Operations inženjeri
- Programeri u timu
- Nepoznati programeri
- Drugi servisi

## Zavisnosti servisa

Vaš servis ima potrebu za:

- hardverom
- softverom
- drugim servisima (baze podataka, servisi koji imaju API)

Softver uključuje:

- druge programe (Python interpreter)
- podatke (sadržaj baza podataka)
- konfiguracije

## Okruženja

Jedan ili više servera na kojima se izvršava aplikacija.

Zavisnosti mogu biti različite u različitim stadijumima razvoja i rada:

- na lokalnom računaru
- na serverima za kompajliranje i pakovanje
- u okruženju za testiranje
- u produkcionim okruženjima

Potrebno je uskladiti rad u različitim okruženjima i omogućiti što lakšu instalaciju, konfigurisanje, održavanje i ažuriranje aplikacije.

# Pakovanje i distribucija

## Pakovanje i distribucija

Svaki sistem definiše svoje konvencije i kreira svoje alate koji omogućavaju objavljivanje softvera i *olakšavaju* korišćenje.

- sistemski paketi (deb, pkg, msi)
- alati programskog jezika (pip, npm, gem)

Python ima bogat spektar vrlo korisnih alata za pakovanje python modula, bilo da su biblioteke i aplikacije. Takodje, pip pruža razrešavanje zavisnosti i instaliranje i kompajliranje svih modula koje koristi vaša biblioteka ili aplikacija.

Problemi:

- kompatibilnost sa različitim verzijama python interpretera
- kompatibilnost izmedju različitih platformi (macOS, linux, windows)
- (ne)dostupnost C/C++ biblioteka
- nekonzistentnost u razrešavanju zavisnosti

## Python virtualenv

Pruža vrlo dobar način izolacije python paketa koji su zavisnosti aplikacije.

Međutim postoje problemi:

- python interpreter i dalje treba da postoji na sistemu
- kreiranje virtualenv-a se obično vrši na host serveru i zahteva instaliranje C/C++ dev bibilioteka
- pakovanje virtualenv-a u poseban paket zahteva dodatno referenciranje potrebnih C/C++ biblioteka koje moraju biti instalirane na host sistemu
- ažuriranjem neke C/C++ biblioteke na verziju koja nije binarno kompatibilna sa onom kojom je kreiran virtualenv zahteva reinstalaciju
- virtualenv se teško premešta sa jedne lokacije na fajl sistemu na drugu

## Linux kontejneri

Izolacija i kontrola resursa Linux procesa na osnovu kernel *namespace* i *cgroups* funkcionalnosti.

Linux kontejner je izolovan proces ili grupa procesa. Procesi su potpuno vidljivi sa host sistema, ali imaju različite proces ID vrednosti u odnosu na kontejner, nemaju pristup host fajl sistemu, već samo izolovanim dalovima fajl sistema, nemaju pristup svim network interfejsima, već samo određenim virtuelnim.

Razlike od koncepta virtualizacije:

- ne zahteva hypervisor
- procesi su izolovani na nivou kernela, ne mogu da se izvrsavaju procesi drugih operativnih sistema ili arhitektura
- nije potrebno instaliranje posebnog operativnog sistema u okviru koga bi se proces izvršavao
- mnogo manji *overhead* izvršavanja

## Prednosti izolacije

Kontejneri omogućavaju da na istom host serveru rade aplikacije koje zahtevaju različite verzije softvera koji im je potreban za rad. Izbegavaju se konflikti između aplikacija, ali i timova.

Host server može da bude vrlo *tanak*, što olakšava održavanje servera.

Moguće je pokrenuti više instanci servisa, u istoj ili različitim verzijama, na istom host serveru, bez obzira kako su ti servisi napisani.

# Docker

## Docker

Alat za kontrolu Linux kontejnera.

CLI program *docker* komunicira sa lokalnim servisom *dockerd* putem HTTP API-a.

Moguće je koristiti docker API i praviti svoje alate i servise koji rade sa kontejnerima. Primer takvog alata je Kubernetes.

Proćićemo kroz koncepte:

- containers
- images
- volumes
- networks

# Instalacija

## Ubuntu Linux

docs.docker.com/engine/installation/linux/ubuntu/#install-docker

(https://docs.docker.com/engine/installation/linux/ubuntu/#install-docker)

## Docker for Mac

docs.docker.com/docker-for-mac/install (https://docs.docker.com/docker-for-mac/install)

## Docker for Windows

docs.docker.com/docker-for-windows/install (https://docs.docker.com/docker-for-windows/install)

# Docker images

Arhiva fajlsistema koji ce biti dostupan kontejneru kada se startuje.

Uključuje i podrazumevane parametre za pokretanje kontejnera.

**Distribucija**

Javno dostupni image-i nalaze se na:

hub.docker.com (https://hub.docker.com)

Za čuvanje privatnih image-a koristi se Docker registry:

docs.docker.com/registry (https://docs.docker.com/registry)

# Pokretanje kontejnera

## Komanda:

```
docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

```
docker run python:3 python --version
```

## Lista kontejnera:

```
docker ps -a
```

## Pokretanje interaktivnog shell-a unutar kontejnera:

```
docker run --rm -ti debian bash
```

## run = create + start

Docker `run` komanda je kombinacija druge dve komande `create` i `start`.

`create` kreira novi kontejner od image-a sa parametrima koji se prosledjuju `run` komandi, a `start` pokrece kontejner, tj. proces koji je definisan da se izvrsi unutar kontejnera.

Kontejner može da izvrši proces i stopira se, ali i dalje postoji.

Kontejner može da ima dugoživeći proces, na primer web server, koji radi u pozadini i mora prvo da se stopira da bi se uklonio.

Ne postoji komanda koja je pokreće `stop` i `rm` kontejnera. Da bi se kontejner potpuno uklonio, nakon `run`-a, potrebno ga je stopirati `stop` i izbrisati `rm`.

Docker `rm -f` nema isto ponašanje kao sukcesivno pozivanje `stop` i `rm`.

# Docker volumes

Način za prosleđivanje podataka sa host sistema kontejneru.

Analogno *mount*-ovanju diska na neku putanju operativnog sistema.

```
echo "print('lokacija skripte', __file__)" > /tmp/script.py

docker run --rm -v /tmp:/app python:3 python /app/script.py
```

Lista postojećih volume-a.

```
docker volume ls
```

Kreiranje volume-a umesto korišćenje putanja.

```
docker volume create radionica-postgres-data
```

# Docker mreže

Mehanizam povezivanja kontejnera putem IP mreza.

## Lista postojećih mreža

```
docker network ls
```

## Kreiranje mreže

```
docker network create radionica
```

# PostgreSQL server u kontejneru

## Startovanje servisa u pozadini unutar kontejnera

```
docker run -d \
    --name radionica-postgres \
    --net radionica \
    --restart unless-stopped \
    -e POSTGRES_USER=radionica \
    -e POSTGRES_PASSWORD=P4ss \
    -v radionica-postgres-data:/var/lib/postgresql/data \
    -p 127.0.0.1:5432:5432 \
    postgres:9.6.2
```

## Zvaničan image:

hub.docker.com/_/postgres (https://hub.docker.com/_/postgres)

```
docker run -t --rm --net radionica debian ping -c 3 radionica-postgres
```

# Docker run opcije

`--name NAME` - davanje posebnog naziva kontejneru
`--rm` - uklanjanje kontejnera nakon sto procesi u njemu završe izvršavanje
`-d` - pokretanje kontejnera u pozadini
`-t` - otvaranje pseudo terminala ka kontejneru
`-i` - otvaranje STDIN I/O stream-a ka kontejneru
`-e KEY=VALUE` - definisanje environment varijable unutar kontejnera
`--net NET` - pokretanje kontejnera unutar željene docker mreže
`-v HOSTPATH:CONTAINERPATH[:OPTIONS]` - mount-ovanje volume-a
`-p [HOSTIP:]HOSTPORT:CONTAINERPORT` - proxiranje saobraćaja
`--restart [no|always|unless-stopped|on-failure]` - restart polisa

docs.docker.com/engine/reference/run (https://docs.docker.com/engine/reference/run)

## Pristupanje PostgreSQL serveru u kontejneru

Komanda `docker exec` izvršava proces unutar aktivnog kontejnera. Ne pokreće novi, kao `run`.

### Konektovanje na bazu

```
docker exec -ti radionica-postgres psql -U radionica
```

### Importovanje SQL podataka

```
docker exec -i radionica-postgres psql -U radionica < radionica.sql
```

### Eksporotovanje SQL podataka

```
docker exec -i radionica-postgres pg_dump -U radionica > radionica_dump.sql
```

## Kreiranje image-a

Dva načina

- docker commit CONTAINER REPOSITORY[:TAG]
- docker build -t REPOSITORY[:TAG] PATH

Komanda docker build zahteva `Dockerfile` čime se omogućava reproducibilnost kreiranja image-a.

## SSH klijent u kontejneru

Korišćenje SSH klijenta pod Windows operativnim sistemom.

Primer kreiranja image-a koristeći commit komandu.

```
docker run --name sshclient debian bash -c "apt-get update && apt-get install -y openssh-client"

docker commit sshclient janos/ssh

docker run --rm -ti janos/ssh ssh -o "StrictHostKeyChecking no" root@server.sedamcvrkuta.com
```

# Primer jednostavne python aplikacije

## primer1/app.py

```
import pg8000
from flask import Flask, Response

app = Flask(__name__)
conn = pg8000.connect(user="radionica", password="P4ss", host="radionica-postgres")
cursor = conn.cursor()
cursor.execute("CREATE TEMPORARY TABLE tweets (id SERIAL, message TEXT)")
cursor.execute("INSERT INTO tweets (message) VALUES (%s), (%s), (%s)", (
    "Don’t Panic.",
    "It is a mistake to think you can solve any major problems just with potatoes!",
    "It is an important and popular fact that things are not always what they seem."
))
conn.commit()

@app.route('/')
def random_tweet():
    cursor.execute("SELECT * FROM tweets ORDER BY RANDOM() LIMIT 1")
    return Response(cursor.fetchall()[0][1], mimetype='text/plain')

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

# Primer jednostavne python aplikacije

## primer1/requirements.txt

```
pg8000==1.10.6
flask==0.12
```

## primer1/Dockerfile

```
FROM python:3

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r requirements.txt

COPY . /usr/src/app

CMD ["python", "app.py"]
```

# Primer jednostavne python aplikacije

## Kreiranje image-a

```
docker build -t janos/radionica-primer1 primer1
```

## Pokretanje kontejnera

```
docker run --rm -d --name primer1 --net radionica -p 5000:5000 janos/radionica-primer1
```

localhost:5000 (http://localhost:5000)

## Stopiranje kontejnera

```
docker stop primer1
```

## Primer gunicorn python aplikacije

primer2/app.py je istog sadržaja kao i u prethodnom primeru.

primer2/requirements.txt

```
pg8000==1.10.6
flask==0.12
gunicorn==19.7.1
```

primer2/Dockerfile

```
FROM python:3

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r requirements.txt
COPY . /usr/src/app

ENV GUNICORN_CMD_ARGS="--bind=0:8000 --worker-class=gthread --threads=10"

CMD ["gunicorn", "app:app"]
```

# Primer gunicorn python aplikacije

## Kreiranje image-a

```
docker build -t janos/radionica-primer2 primer2
```

## Pokretanje kontejnera

```
docker run --rm -d --name primer2 --net radionica -p 8000:8000 janos/radionica-primer2
```

localhost:8000 (http://localhost:8000)

## Stopiranje kontejnera

```
docker stop primer2
```

# Nginx u kontejneru

- proxy štiti python wsgi server od malicioznih upita
- može da terminira SSL konekcije

```
server {
    access_log /var/log/nginx/primer2-access.log;
    error_log /var/log/nginx/primer2-error.log;

    location / {
        proxy_pass http://primer2:8000;
    }
}
```

```
docker run -d --rm \
    --name primer2-nginx \
    --net radionica \
    -v $(pwd)/primer2/primer2.conf:/etc/nginx/conf.d/default.conf \
    -p 9000:80 \
    nginx
```

localhost:9000 (http://localhost:9000)

## Instaliranje sistemskih paketa

Neki python moduli zahtevaju C/C++ biblioteke da bi se instalirali.

Takođe, SSL CA sertifikati su potrebni ako HTTP klijenti komuniciraju preko enkriptovanog HTTPS protokola.

```
FROM python:3

RUN apt-get update && \
    apt-get install -y ca-certificates libyaml-dev && \
    rm -rf /var/lib/apt/lists/*

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY . /usr/src/app

RUN pip install --no-cache-dir -r requirements.txt

CMD ["python", "app.py"]
```

# Docker image push

Upload docker image-a na javni ili privatni registry.

Javan registry:

hub.docker.com (https://hub.docker.com)

Nakon registracije, potrebno je prijaviti se:

```
$ docker login
Username (janos):
Password:
```

Push image-a:

```
docker push janos/radionica-primer2
```

# Konfigurisanje aplikacije u kontejneru

- environment varijablama

Komanda docker run moze da prihvati visestruke `-e` parametre:

```
docker run -e SEVENTWEETS_DB_HOST=pghostname -e SEVENTWEETS_DB_USERNAME=seventweets ...
```

- mount-ovanjem volume-a sa konfiguracionim fajlovima

Konfiguracije mogu da se čuvaju na host fajlsistemu i da se proslede kontejenru:

```
docker run -v /srv/config/seventweets:/config:ro ...
```

- trećim servisom (consul, etcd...)

# Kontejner log i diff

```
docker logs CONTAINER
```

Lista log poruke postojećeg kontejnera.

```
docker logs radionica-postgres
```

```
docker diff CONTAINER
```

Lista izmenjene fajlove u kontejneru u odnosu na image.

```
docker diff radionica-postgres
```

fabric

# fabric

```
pip install fabric3
```

Automatizacija izvršavanja komandi preko SSH ili na lokalnom računaru.

Python biblioteka `fabric` i CLI komanda `fab`.

Svi taskovi se definišu u `fabfile.py` fajlu koji `fab` komanda učitava i izvršava.

www.fabfile.org (http://www.fabfile.org)

Vrlo dobre alternative:

- ansible

  www.ansible.com (https://www.ansible.com)

- salt stack

  saltstack.com/community (https://saltstack.com/community)

# fabric

## fabfile.py

```
from fabric.api import run

def serve():
    run("docker run -d -v /usr/share/doc:/doc -w /doc -p 8000:8000 python python -m http.server")
```

## Izvršavanje taska:

```
fab -H server.sedamsvrkuta.com serve
```

## fabric.api.run i fabric.api.local

Dve funkcije kojima se izvršavaju komande.

Funkcija `run("komanda")` izvršava komandu preko SSH protokola.

Funkcija `local("komanda")` izvršava komandu u lokalnom terminalu.

```
from fabric.api import run, local

def uname_local():
    local("uname -a")

def uname_remote():
    run("uname -a")
```

# fabric.api.env

Dictionary koji sadrži razne informacije vezane za izvršavanje taskova.

Najbitnije vrednosti su lista servera

```
fabric.api.env.hosts = ['moj-server.com']
```

SSH korisnik

```
fabric.api.env.user = 'ja'
```

oznaka za nastavljanje izvršavanja i nakon greške

```
fabric.api.env.warn_only = False
```

# fabric.api.settings

Moguće je dodeliti vrednost nekoj varijabli u samo određenom delu koda na sledeći način:

```
from fabric.api import run, settings

def start():
    with settings(warn_only=True):
        run("false")
    run("true")
```

## Argumenti taskova

Task funkcija može da sadrži argumente kako bi se prosledili opcioni paramteri pri izvršavanju.

```
from fabric.api import run

def serve(path=""):
    if path is "":
        path = "/usr/share/doc"
    run("docker run -d -v "+path+":/doc' -w /doc -p 8000:8000 python python -m http.server")
```

Izvršavanje taska sa opcionom putanjom:

```
fab -H server.sedamsvrkuta.com serve:/tmp
```

# primer2 fabfile.py

```
from fabric.api import local, settings, run, env

# Lista servera
env.hosts = ["server.sedamcvrkuta.com"]
# SSH korisnik
env.user = "root"

# Varijable servisa
name = "primer2"
port = 8000
repository = "janos/radionica-primer2"
network = "radionica"
pg_user = "radionica"
pg_pass = "P4ss"
pg_port = 5432
pg_version = "9.6.2"
pg_volume = "radionica-postgres-data"
```

# primer2 fabfile.py

```
def build(tag=""):
    if tag is not "":
        tag = ":" + tag
    local(f"docker build -t {repository}{tag} .")


def push(tag=""):
    local(f"docker push {repository}{tag}")


def create_network():
    local(f"docker network create {network}")
```

# primer2 fabfile.py

```
def start(tag=""):
    local(f"""
        docker run -d \
            --name {name} \
            --net {network} \
            -p {port}:{port} \
            {repository}{tag}
        """)


def stop():
    local("docker stop {}".format(name))
    local("docker rm {}".format(name))


def restart():
    start()
    stop()
```

# primer2 fabfile.py

```
def db_start():
    with settings(warn_only=True):
        local(f"docker volume create {pg_volume}")
    local(f"""
        docker run -d \
            --name radionica-postgres \
            --net {network} \
            --restart unless-stopped \
            -e POSTGRES_USER={pg_user} \
            -e POSTGRES_PASSWORD={pg_pass} \
            -v {pg_volume}:/var/lib/postgresql/data \
            -p 127.0.0.1:5432:5432 \
            postgres:{pg_version}
        """)


def db_stop():
    local("docker stop radionica-postgres")
    local("docker rm radionica-postgres")
```

# primer2 fabfile.py

```
def deploy_db():
    with settings(warn_only=True):
        run(f"docker volume create {pg_volume}")
        run(f"docker network create {network}")
    run(f"""
        docker run -d \
            --name radionica-postgres \
            --net {network} \
            --restart unless-stopped \
            -e POSTGRES_USER={pg_user} \
            -e POSTGRES_PASSWORD={pg_pass} \
            -v {pg_volume}:/var/lib/postgresql/data \
            postgres:{pg_version}
        """.)
```

# primer2 fabfile.py

```
def deploy(tag=""):
    build(tag)
    push(tag)
    with settings(warn_only=True):
        run(f"docker stop {name}")
        run(f"docker rm {name}")
        run(f"docker network create {name}")
    run(f"""
        docker run -d \
            --name {name} \
            --net {network} \
            -p {port}:{port} \
            {repository}{tag}
        """)
```

# Travis CI

## Travis CI

Automatizovano testiranje i deploy projekata hostovanih na GitHub-u.

travis-ci.org (https://travis-ci.org)

Jednostavno povezivanje GitHub naloga i Travis-a.

Besplatno za open source projekte. Sadrži i komercijalnu varijantu za privatne GitHub projekte.

Sistem sa velikim brojem mogućnosti i vrlo dobrom dokumentacijom.

Deklarativan način konfigurisanja izvršavanja testova kroz fajl `.travis.yml` u okviru git repoa.

Testovi se pokrecu nakon svakog commita.

# Primer

## Fajl .travis.yml

```
language: python
python:
    - "3.6"
install:
    - pip install -r requirements.txt
script:
    - pytest
```

# Testiranje sa bazom podataka

## Fajl .travis.yml

```
language: python
python:
    - "3.6"
env:
    - DB_HOST=localhost
    - DB_PORT=5432
    - DB_USER=postgres
    - DB_PASS=
    - DB_NAME=primer2
services:
    - postgresql
before_script:
    - psql -c 'create database primer2;' -U postgres
install:
    - pip install -r requirements.txt
script:
    - pytest
```

# Thank you

Janoš Guljaš
Seven Bridges Genomics
janos.guljas@sbgenomics.com (mailto:janos.guljas@sbgenomics.com)
@janosguljas (http://twitter.com/janosguljas)