Part 2

[slide 10]

(you can copy the files from part 1 to another folder, and start from there)

Let's build a web app.

Go to http://flask.pocoo.org/!

```
$ pip install Flask
```

Change `app.py` to:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

app.run(host="0.0.0.0", debug=True)
```

Run the app and check http://0.0.0.0:5000/:

```
$ python app.py
```

Remove the `sleeper` from `docker-compose.yml`:

```
version: "3"
services:
  app:
    build: .
```

Run `docker-compose up --build` and check http://0.0.0.0:5000/.

It doesn't work. Why?

We need to expose and publish the container's port `5000` to the host (our machine):

```
version: "3"
services:
  app:
    build: .
    ports:
      - "3333:5000"
```

The above publishes container's port `5000` on host's port `3333`.

Run `docker-compose up` and http://0.0.0.0:3333/.

Deploy it on Kubernetes

[slide 11]

Create file `app.yml` (simply based on Kubernetes 101):

```
apiVersion: v1
kind: Pod
metadata:
  name: app
  labels:
    foo: vitor
spec:
  containers:
  - name: app
    image: vitorenesduarte/tutorial
```

(Compared to 101, we added `foo: vitor` as a label, because `kubectl expose`, which we will use later, requires it)

And deploy it on Kubernetes.

(for that you need `CONFIG`, a Kubernetes configuration file, which I will provide)

```
$ kubectl --kubeconfig=CONFIG create -f app.yml
```

QUESTION: will this work?

```
$ kubectl --kubeconfig=CONFIG get pods
NAME      READY     STATUS         RESTARTS   AGE
app       0/1       ErrImagePull   0          4s
```

Ups. The docker image is still local.

Let's push it to Docker Hub.

Create an account there, and login with `docker login`. Then:

```
$ docker build -t vitorenesduarte/tutorial .
$ docker push vitorenesduarte/tutorial
```

Before anything else, let's avoid always having to specify `--kubeconfig`.

Let's check the manual.

```
$ kubectl config --help | sed -n '5,7p'
```

We can simply have `$KUBECONFIG` environment variable pointing to the `CONFIG` file, e.g.:

```
$ export KUBECONFIG=$(pwd)/CONFIG
```

Now, let's delete the `app` pod and deploy again.

```
$ kubectl delete pod app
$ kubectl get pods
$ kubectl create -f app.yml
$ kubectl get pods --watch
```

Some magic so that we can access our app:

```
$ kubectl expose -f app.yml --type=LoadBalancer --port 5000 --target-port=3333
$ kubectl get service app
NAME      TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
app       LoadBalancer   10.7.241.40   <pending>     5000:31171/TCP   53s
```

Watch until `EXTERNAL-IP` is no longer ‘:

```
$ kubectl get service app --watch
```

And then go to IP:3333.

TODO. Add an id to the app, and show load balancing working.