Julia in production environments

Sebastian Zając, PhD

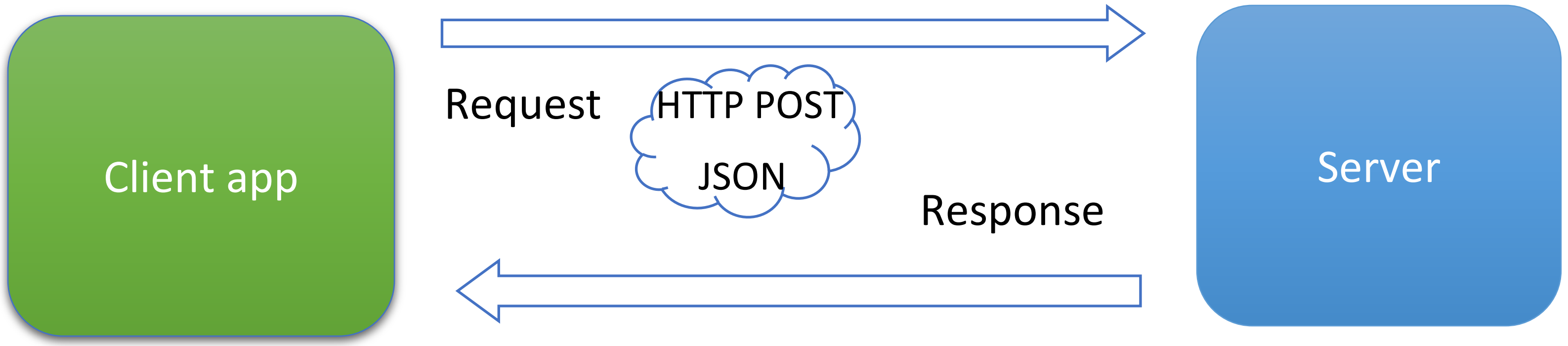
SGH Warsaw School of Economics

Preparation of this workshop has been supported by the Polish National Agency for Academic Exchange under the Strategic Partnerships programme, grant number BPI/PST/2021/1/00069/U/00001.

Business case

Let’s assume you work for company that sells apartments in Boston.
We want to increase our sales and offer better quality services to our customers by new mobile application which can be used even by 1 000 000 people simultaneously.

We can realize this by serving a prediction of house value when the user sends a request for pricing over the web.



On a server we will run a machine learning model that is trained to predict house price based on its features. We will use the Housing data set from UCI repository to build it.

*https://archive.ics.uci.edu/ml/machine-learning-databases/housing/.

What exactly is Serving a Model?

- Training a **good ML** model is **ONLY** the first part:
  You do need to make your model available to your end-users
  You do this by either providing access to the model on your SERVER.
- **When serving ML Model You need:**
  a model, an interpreter, input data.
- **Important Metrics:**
  1. Latency,
  2. Cost,
  3. Throughput (number of requests served per unit time)

REST API

Sharing data between two or more systems has always been a fundamental requirement of software development – DevOps vs MLOps

REST: *Representational State Transfer* **API**: *Application Programming Interfaces*

- **Client-Server:** Client (systemA) makes an **HTTP request** to a **URL** hosted by SystemB, which returns a **response**. It's identical to how a browser works. A browser makes a request for a specific URL. The request is routed to a web server which typically returns an HTML (text) page.
- **Stateless:** The client request should contain all the information necessary to respond.

Request

- An Endpoint URL a domain, port, path, query string	http://mydomain:8000/getapi?&val1=43&val2=3
- The HTTP methods	GET, POST
- HTTP headers contain authentication information, cookies	Content-Type: application/json, text … Accept: application/json Authorization: Basic abase64string, Tokens etc

Request body

json example

```
{
        "RAD": 1,
        "PTRATIO": 15.3,
        "INDUS": 2.31,
        "B": 396.9,
        "ZN": 18,
        "DIS": 4.09,
        "CRIM": 0.00632,
        "RM": 6.575,
        "AGE": 65.2,
        "CHAS": 0,
        "NOX": 0.538,
        "TAX": 296,
        "LSTAT": 4.98
}
```

Response

The **response** payload can be whatever is practical:

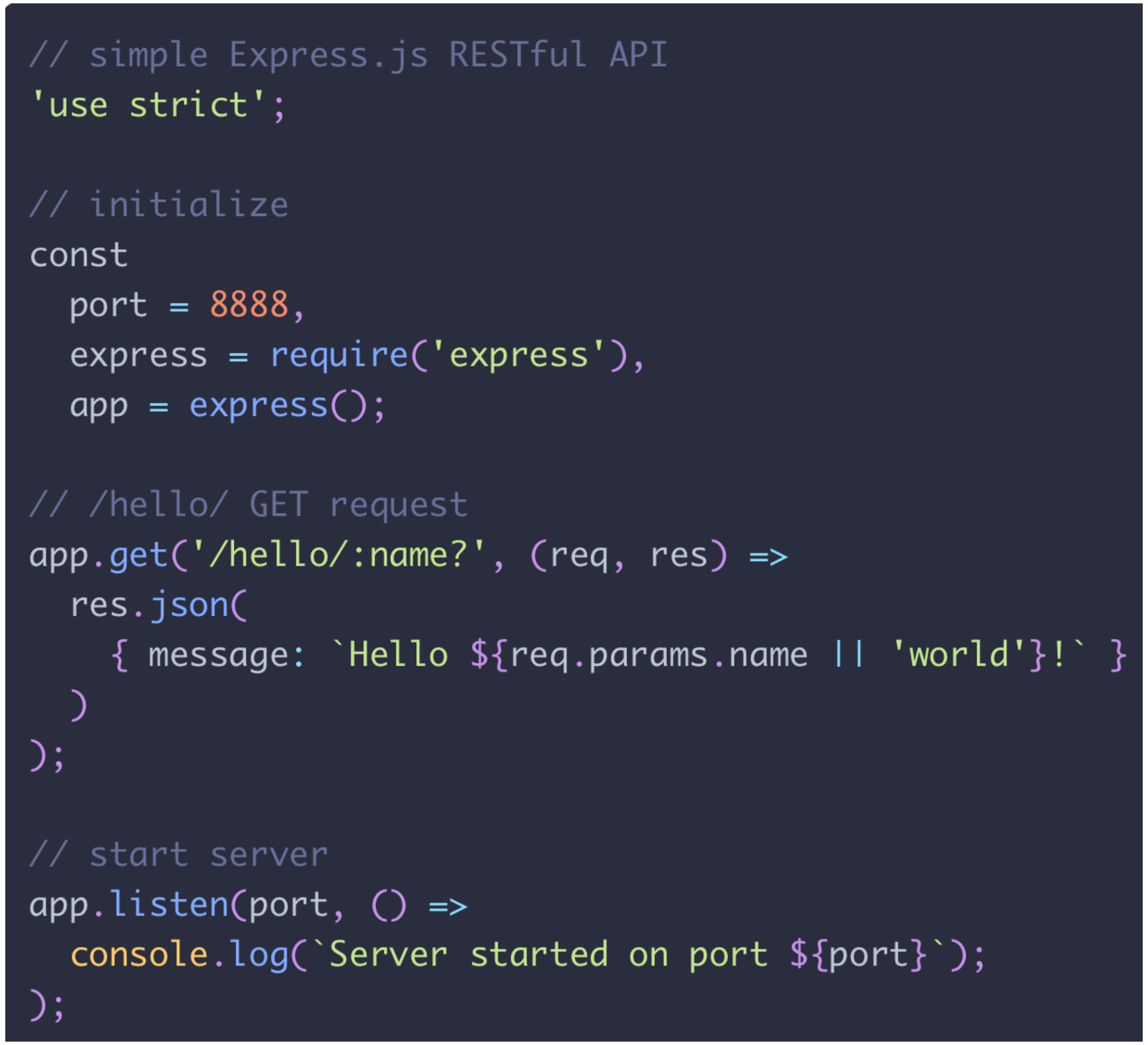
- data,
- HTML,
- an image,
- an audio file,
- and so on.

Data responses are typically JSON-encoded, but XML, CSV, simple strings, or any other format can be used.

HTTP status code:

- 200 OK is used for successful requests,
- 400 Bad Request,
- 404 Not Found,
- 401 Unauthorized

REST API examples

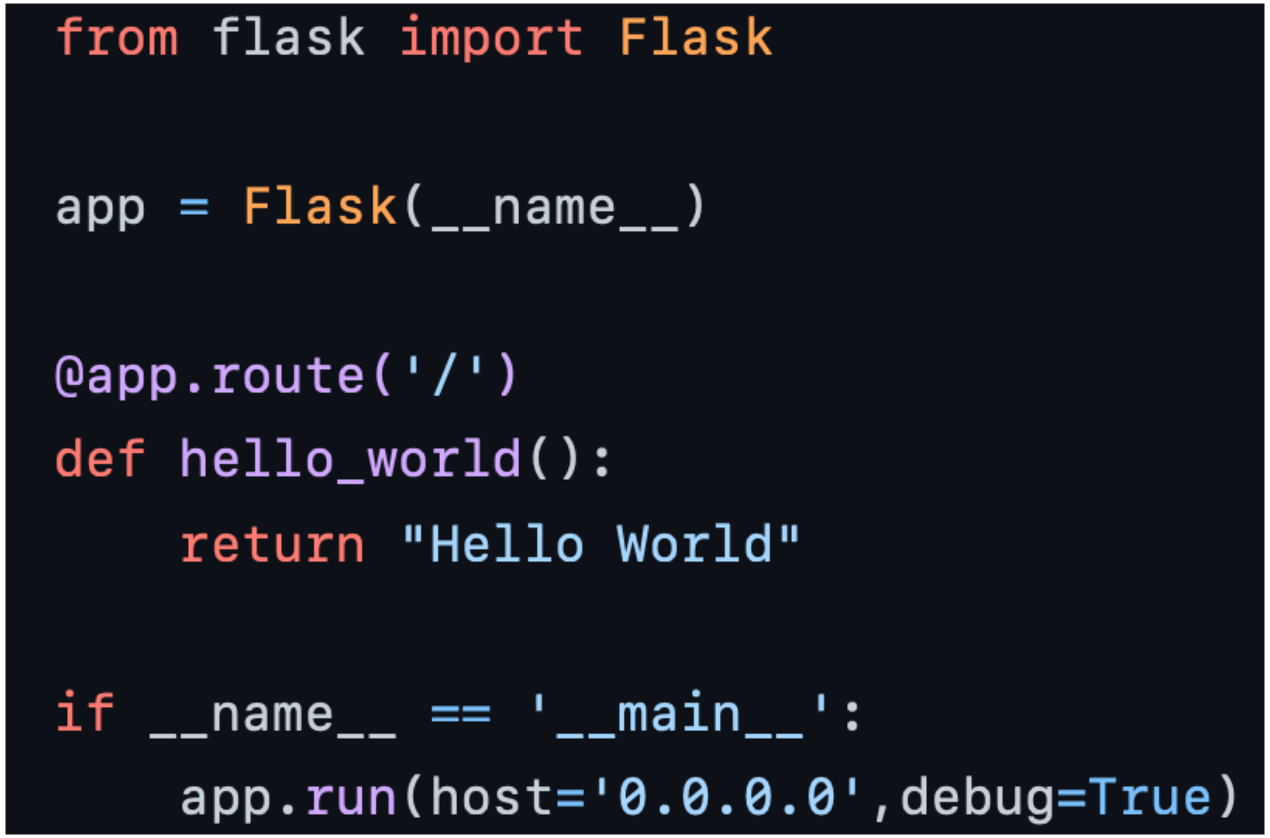


```javascript
// simple Express.js RESTful API
'use strict';

// initialize
const
  port = 8888,
  express = require('express'),
  app = express();

// /hello/ GET request
app.get('/hello/:name?', (req, res) =>
  res.json(
    { message: `Hello ${req.params.name || 'world'}!` }
  )
);

// start server
app.listen(port, () =>
  console.log(`Server started on port ${port}`);
);
```



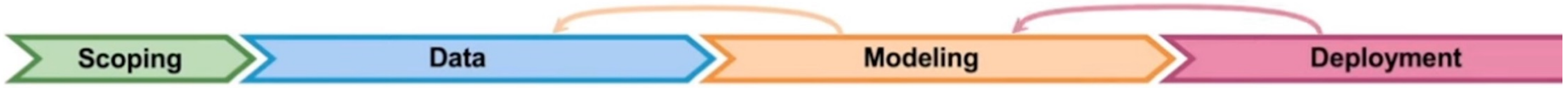
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello World"

if __name__ == '__main__':
    app.run(host='0.0.0.0',debug=True)
```

The ML project lifecycle



- ✓ Define project
- ✓ Business points

- ✓ Define data
- ✓ Label and organize data

- ✓ Select and train model
- ✓ Perform error analysis

- ✓ Deploy in production
- ✓ Monitor & maintain system

The ML project lifecycle

D. Sculley et All. *Hidden Technical Debt in Machine Learning Systems* (2015)

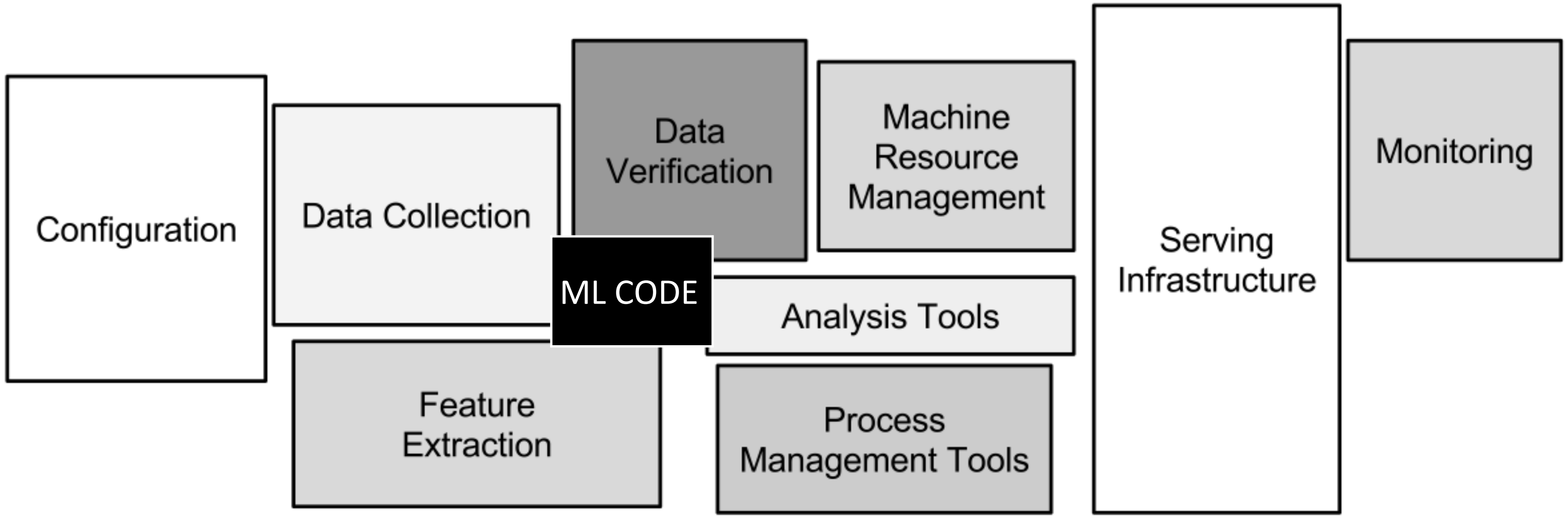


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Docker

Docker *is an amazing tool that allows you to ship your software along with all of its dependencies.*

1. When You create Your model first you setup environment on your local machine. But now you want to share this model with a colleague of yours who does not have Python installed. In a pre-Docker time your colleague would have to install all of this software just to run the model.
2. Other solutions? Virtual Machine - why not ?
3. Simple Installation - Docker Desktop (Windows, Mac OS, Linux) - with Kubernetes

Docker

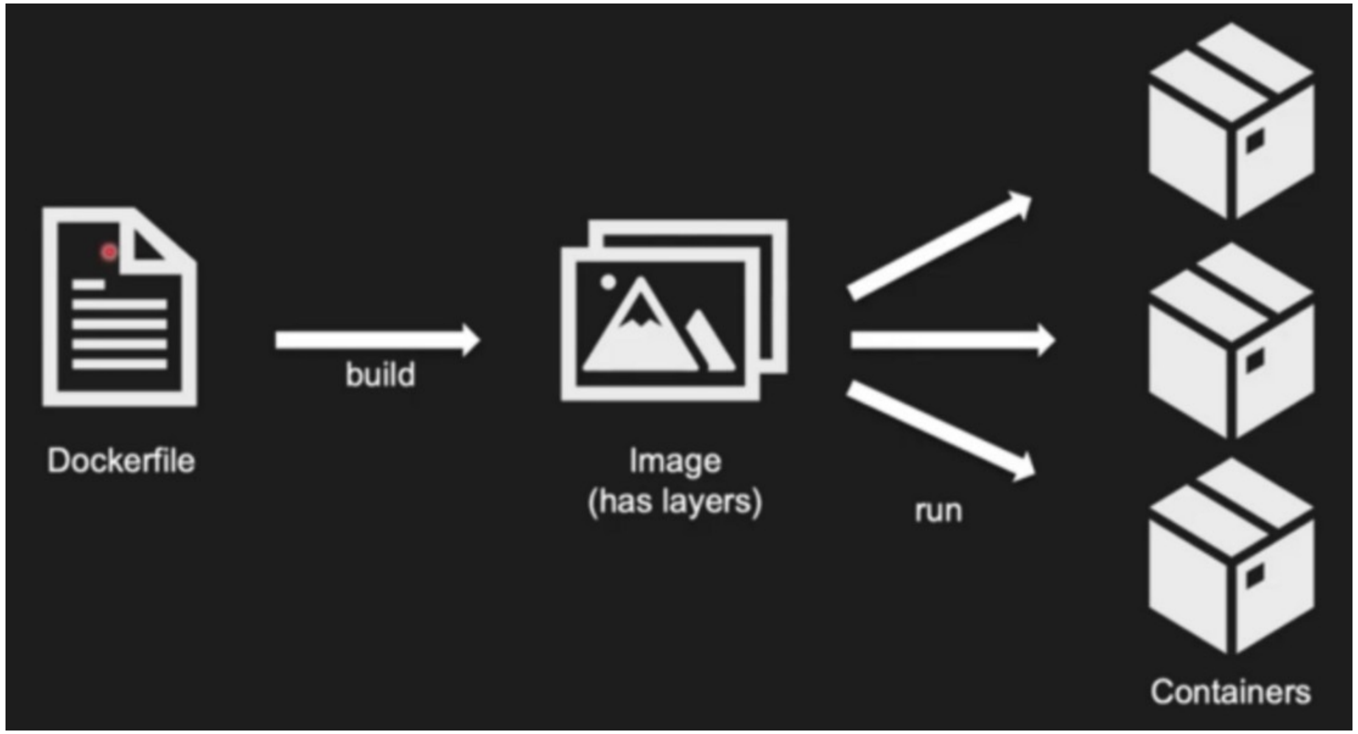
IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

Docker components

- **Dockerfile:** this is special file that contains all of the instructions required to build an image. Like with terminal commands.
- **Image:** This refers to the collection of all your software in one single place. In our case study image will include Julia env and packages like Genie, GLM, DataFrames… Image saving the information of your code and its dependencies.
- **Container**: This is running instance of an image. Usually meant to perform a single task

Running Docker



1. Prepare Dockerfile – ordinary text file
2. Build it and create image: *docker biuld –t name .*

You can run as many containers as you want, even on the one machine