Deployment, Hosting and Monitoring

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#java-web

Deployment

What is Deployment?

- **Deployment** means to push changes or updates from one environment to another

Where to Deploy?

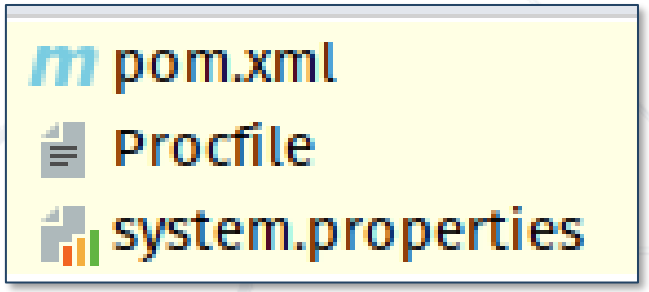
- We can deploy **one project** onto **multiple websites**
- Some of the deployment websites
  - **Heroku**
  - **Amazon Web Services (AWS)**
  - **Google Cloud Platform**

Deploying On Heroku

- There are **3 ways** to deploy a project on Heroku
  - Using **Git** (Heroku Git, Heroku CLI)
  - Using **Github**
  - Using the **Container Registry** (Heroku CLI)

Procfile and System.properties

- Before running our project, we should add **3 important keys** to deploy the project
- Create 2 new files in our **project folder**
  - `Procfile`
  - `system.properties`



System.properties

- system.properties
  - Holds **all of the system configuration properties** needed to run the project
  - By default, Heroku uses JDK Version 1.8
  - To specify specific version:

```
java.runtime.version={version}
```

Procfile

- Procfile
  - Holds the executed commands by the application on startup
  - Should include:

```
web: java -jar target/{name}-{version}.jar
```

pom.xml

```
<version>1.0.0-SNAPSHOT</version>
<name>project_name</name>
```

Application.properties

application.properties

```
spring.datasource.url=${JDBC_DATABASE_URL:}
spring.datasource.username=${JDBC_DATABASE_USERNAME:}
spring.datasource.password=${JDBC_DATABASE_PASSWORD:}

server.port=${PORT:8080}

spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.maximum-pool-size=10

spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.PostgreSQLDialect
```

Managing config vars (1)

- Using the Heroku CLI
  - View current config var values

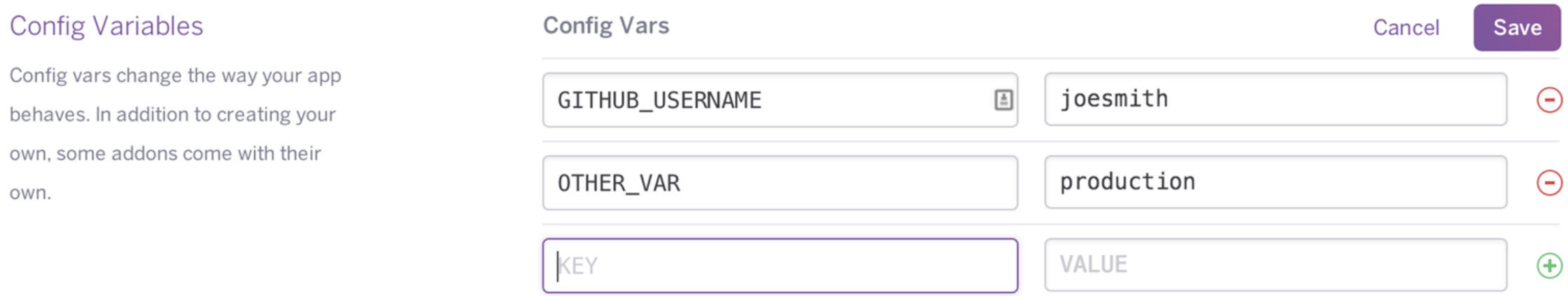
```
$ heroku config
GITHUB_USERNAME: ivan
OTHER_VAR: student
$ heroku config:get GITHUB_USERNAME
ivan
```

  - Set a config var

```
$ heroku config:set SOME_SECRET = mySecret
Adding config vars and restarting myapp... done, v12
SOME_SECRET: mySecret
```

Managing config vars (2)

- Using the Heroku Dashboard



- Using the Heroku Dashboard
  - You can manage your app's config vars programmatically with the Heroku Platform API using a simple HTTPS REST client and JSON data structures

Deploy with Git

Deploying On Heroku with Git (1)

- Download **Heroku CLI**
- In the bash terminal, write the command

```
heroku login
```

- For creating a new Git repository
  - Go to the directory of the project
  - In the bash terminal, write

```
git init
```

Deploying On Heroku with Git (2)

- Create a new Git repository

```
git add .
git commit
```

- Create a new Heroku project and bind it with the git repository

```
heroku create
git remote -v
heroku git: remote -a <project-name>
```

Deploying On Heroku with Git (3)

- Add the PostgreSQL addon

```
heroku addons:create heroku-postgresql
```

- Push the project to Heroku

```
git push heroku master
```

- Change the **ps:scale**

```
heroku ps:scale web=1
```

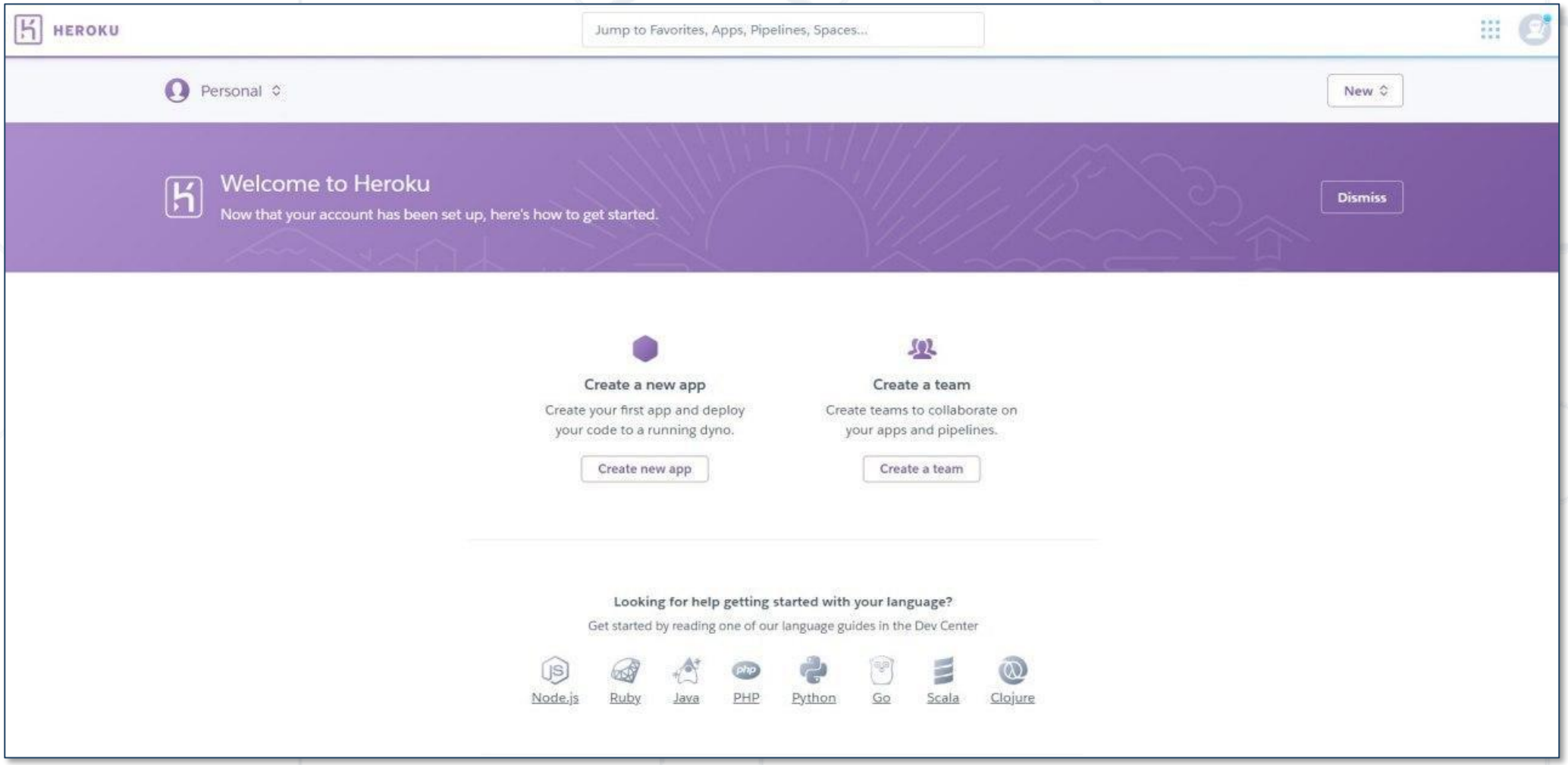
- Check logs

```
heroku logs --tail
```

Deploy with Github

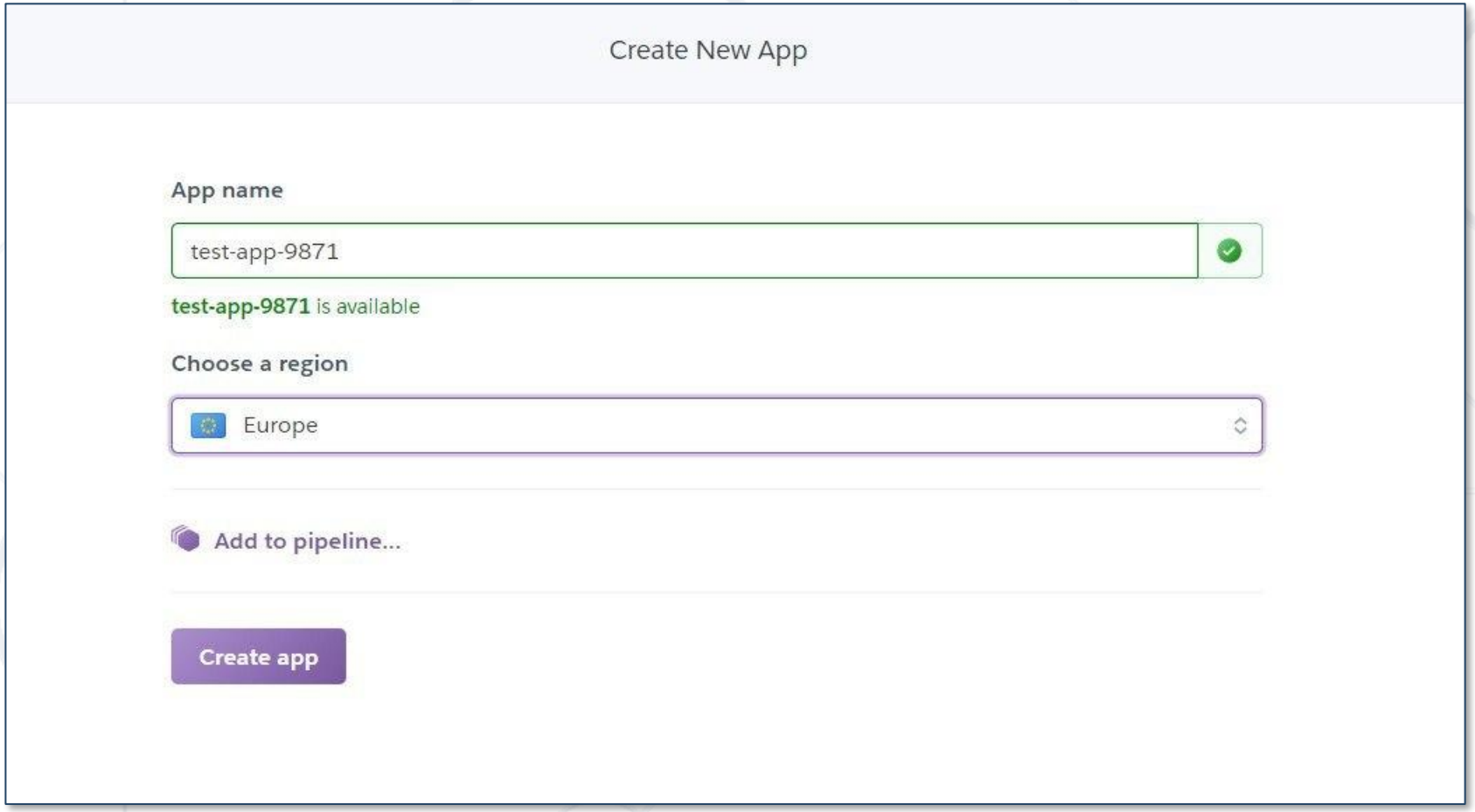
Deploying On Heroku with Github (1)

- Create a registration in the Heroku website



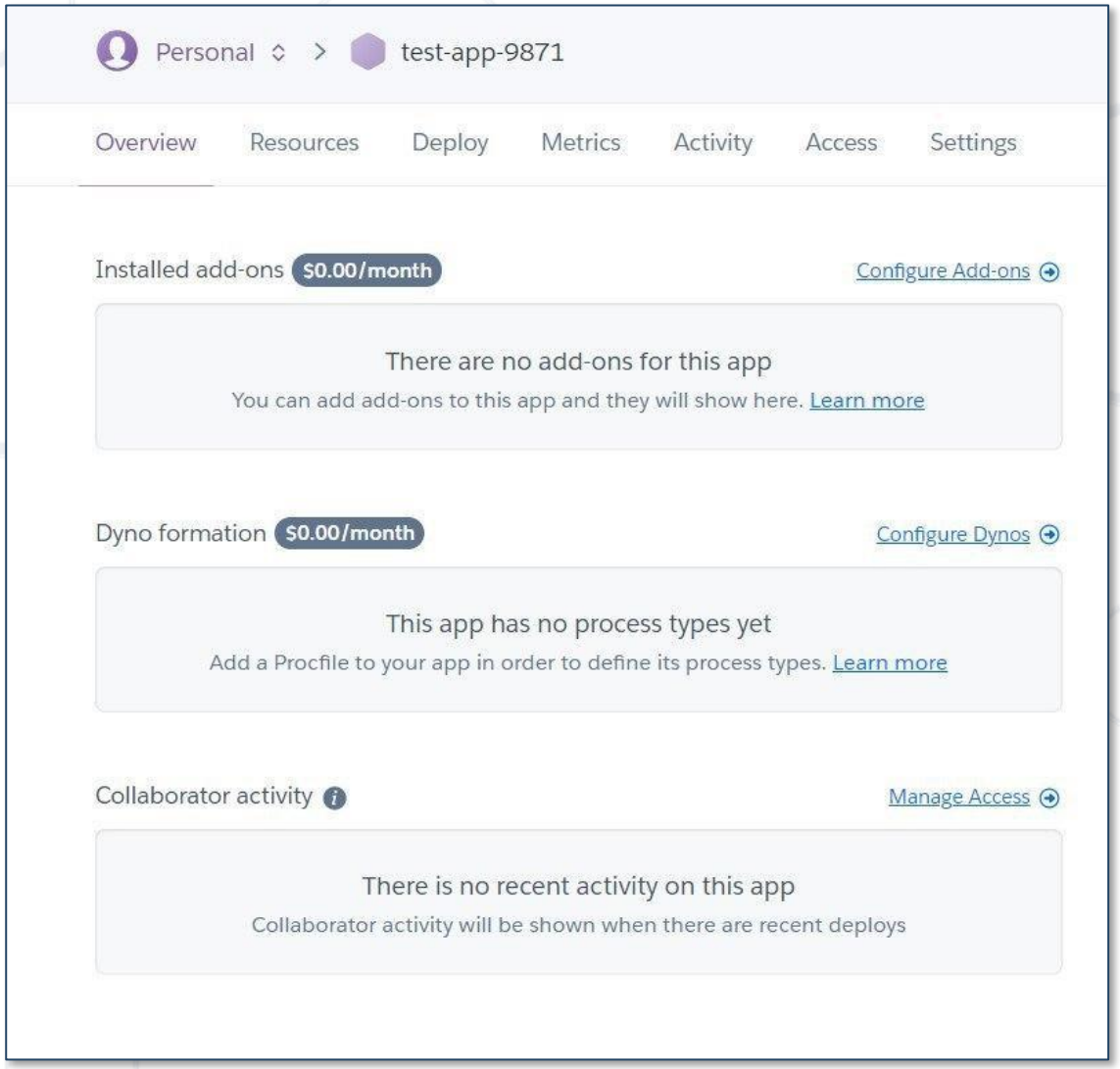
Deploying On Heroku with Github (2)

- Create a new App



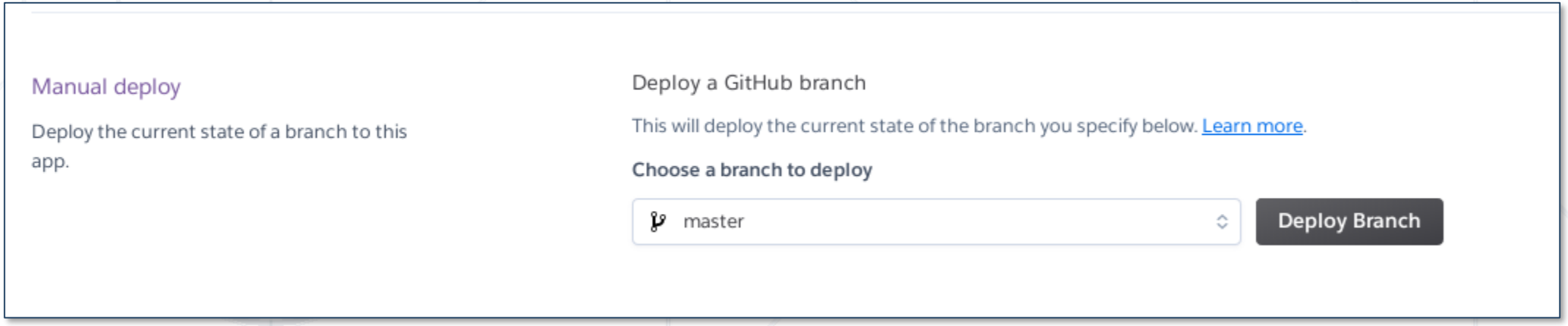
Deploying On Heroku with Github (3)

- Add to the installed add-ons:
  - **heroku-postgres**



Deploying On Heroku with Github (4)

- Go to deploy tab, check "Github" and add your repository



Actuator

Monitor and manage your application

Actuator

- Spring Boot includes a number of **additional features** to help you **monitor** and **manage** your application when you push it to production
- You can choose to manage and monitor your application by using **HTTP endpoints** or with **JMX**
- Auditing, health, and metrics gathering can also be **automatically applied** to your application

Actuator dependency

- The recommended way to enable the features is to add a dependency on the spring-boot-starter-actuator 'Starter'.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
</dependencies>
```

Actuator Endpoints

- Endpoints let you monitor and interact with your application
- Spring Boot includes a number of **built-in endpoints** and lets you add your own
- Each individual endpoint can be enabled or disabled and exposed

Actuator example

- For example, by default, the health endpoint is mapped to **/actuator**

localhost:8080/actuator


```
{"_links":{"self":{"href":"http://localhost:8080/actuator","templated":false},"health":
{"href":"http://localhost:8080/actuator/health","templated":false},"health-path":
{"href":"http://localhost:8080/actuator/health/{*path}","templated":true},"info":
{"href":"http://localhost:8080/actuator/info","templated":false}}}
```

Expose all actuator endpoints

- To expose **all** actuator **endpoints** you need to add in application.properties file:

management.endpoints.web.exposure.include=*

localhost:8080/actuator


```
{"_links":{"self":{"href":"http://localhost:8080/actuator","templated":false},"beans":
{"href":"http://localhost:8080/actuator/beans","templated":false},"caches-cache":
{"href":"http://localhost:8080/actuator/caches/{cache}","templated":true},"caches":
{"href":"http://localhost:8080/actuator/caches","templated":false},"health":
{"href":"http://localhost:8080/actuator/health","templated":false},"health-path":
{"href":"http://localhost:8080/actuator/health/{*path}","templated":true},"info":
{"href":"http://localhost:8080/actuator/info","templated":false},"conditions":
{"href":"http://localhost:8080/actuator/conditions","templated":false},"configprops":
{"href":"http://localhost:8080/actuator/configprops","templated":false},"env":
{"href":"http://localhost:8080/actuator/env","templated":false},"env-toMatch":
{"href":"http://localhost:8080/actuator/env/{toMatch}","templated":true},"loggers":
{"href":"http://localhost:8080/actuator/loggers","templated":false},"loggers-name":
{"href":"http://localhost:8080/actuator/loggers/{name}","templated":true},"heapdump":
{"href":"http://localhost:8080/actuator/heapdump","templated":false},"threaddump":
{"href":"http://localhost:8080/actuator/threaddump","templated":false},"metrics-requiredMetricName":
{"href":"http://localhost:8080/actuator/metrics/{requiredMetricName}","templated":true},"metrics":
{"href":"http://localhost:8080/actuator/metrics","templated":false},"scheduledtasks":
{"href":"http://localhost:8080/actuator/scheduledtasks","templated":false},"mappings":
{"href":"http://localhost:8080/actuator/mappings","templated":false}}}
```

Enabling Endpoints

- If you prefer all endpoints to be disabled
  - Set the **management.endpoints.enabled-by-default = false**
- Use individual endpoint enabled properties
  - On example, enable info endpoint

```
management.endpoints.enabled-by-default=false
management.endpoint.info.enabled=true
```

Securing HTTP Endpoints

- You should take care to **secure** HTTP **endpoints** in the same way that you would any other sensitive URL
- If Spring Security is present, endpoints are **secured by default**
- Example of **custom** security **configuration** for HTTP endpoints

```
@Configuration()
public class ActuatorSecurity extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requestMatcher(EndpointRequest.toAnyEndpoint()).authorizeRequests((requests) ->
                requests.anyRequest().hasRole("ROLE_ADMIN"));
        ...
    }}
```

Implementing Custom Endpoints (1)

- If you add a **@Bean** annotated with **@Endpoint**, any methods annotated with **@ReadOperation**, **@WriteOperation**, or **@DeleteOperation** are automatically exposed over JMX and, in a web application, over HTTP

```
@Component
@Endpoint(enableByDefault = true, id="custom")
public class CustomEndpoint {
    @ReadOperation
    public String getMyEndpoint(){
        return "My custom endpoint";
    } }
```

Implementing Custom Endpoints (2)

- If we want we can create Endpoints with **@RestControllerEndpoint** annotation

```
@Component
@RestControllerEndpoint( id="myRestEndpoint" )
public class MyRestEndpoint {

    @GetMapping("/test")
    @ResponseBody
    public String test(){
        return "My custom rest endpoint";
    }
}
```

Customizing properties

- Customizing the Management Endpoint Paths
  - management.**endpoints.web.base-path**=/manage
- Customizing the Management Server Port
  - management.**server.port**=8081
- Disabling HTTP Endpoints
  - management.**server.port**=-1

Vizualization Tools

- Using the Spring Boot Actuator give us a lot of **information** our application, but it's **not** very **user-friendly**
- Can be integrated with **Spring Boot Admin** for visualization, but it has it's limitations and it's less popular
- Tools like **Prometheus** and **Grafana** are more commonly used for the monitoring and visualization and are language/framework-independent
  - These tools have their own set of data formats and converting the metrics data

Micrometer

Micrometer

- Solves the problem of being a **vendor-neutral data** provider
- Automatically **exposes** `/actuator/metrics` **data** into something your monitoring system can **understand**
- You need to include a vendor-specific micrometer dependency

Micrometer Dependency

- Micrometer is a separate open-sourced project and is not in the Spring ecosystem, so we have to explicitly add it as a dependency
- If using Prometheus, add it's **specific** dependency

```
<dependency>
      <groupId>io.micrometer</groupId>
      <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Micrometer Example

- After adding the micrometer dependency, we have a **new endpoint** - /actuator/**prometheus**
- The data is formatted in **specific** for Prometheus **format**

localhost:8080/actuator/prometheus

```
# HELP process_cpu_usage The "recent cpu usage" for the Java Virtual Machine process
# TYPE process_cpu_usage gauge
process_cpu_usage 2.560625718003065E-4
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 1.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.6256954
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/",} 2.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/",} 0.0566226
# HELP http_server_requests_seconds_max
# TYPE http_server_requests_seconds_max gauge
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 0.0
http_server_requests_seconds_max{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/",} 0.0
# HELP jvm_classes_loaded_classes The number of classes that are currently loaded in the Java virtual machine
# TYPE jvm_classes_loaded_classes gauge
jvm_classes_loaded_classes 7273.0
# HELP jvm_threads_live_threads The current number of live threads including both daemon and non-daemon threads
# TYPE jvm_threads_live_threads gauge
jvm_threads_live_threads 28.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 3145728.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 2.62144E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 3.2555008E7
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-nmethods'",} 2555904.0
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 2.62144E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 4849664.0
jvm_memory_committed_bytes{area="nonheap",id="CodeHeap 'non-profiled nmethods'",} 7077888.0
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge
```

Prometheus

Prometheus

- Time-series database that **stores** the **metric** data by pulling it (using a built-in data scraper) **periodically** over HTTP
- Intervals between pulls can be configured
- Has a simple user interface where we can **visualize**/**query** on all of the **collected metrics**
- To configure Prometheus more precisely we using the **prometheus.yaml** file

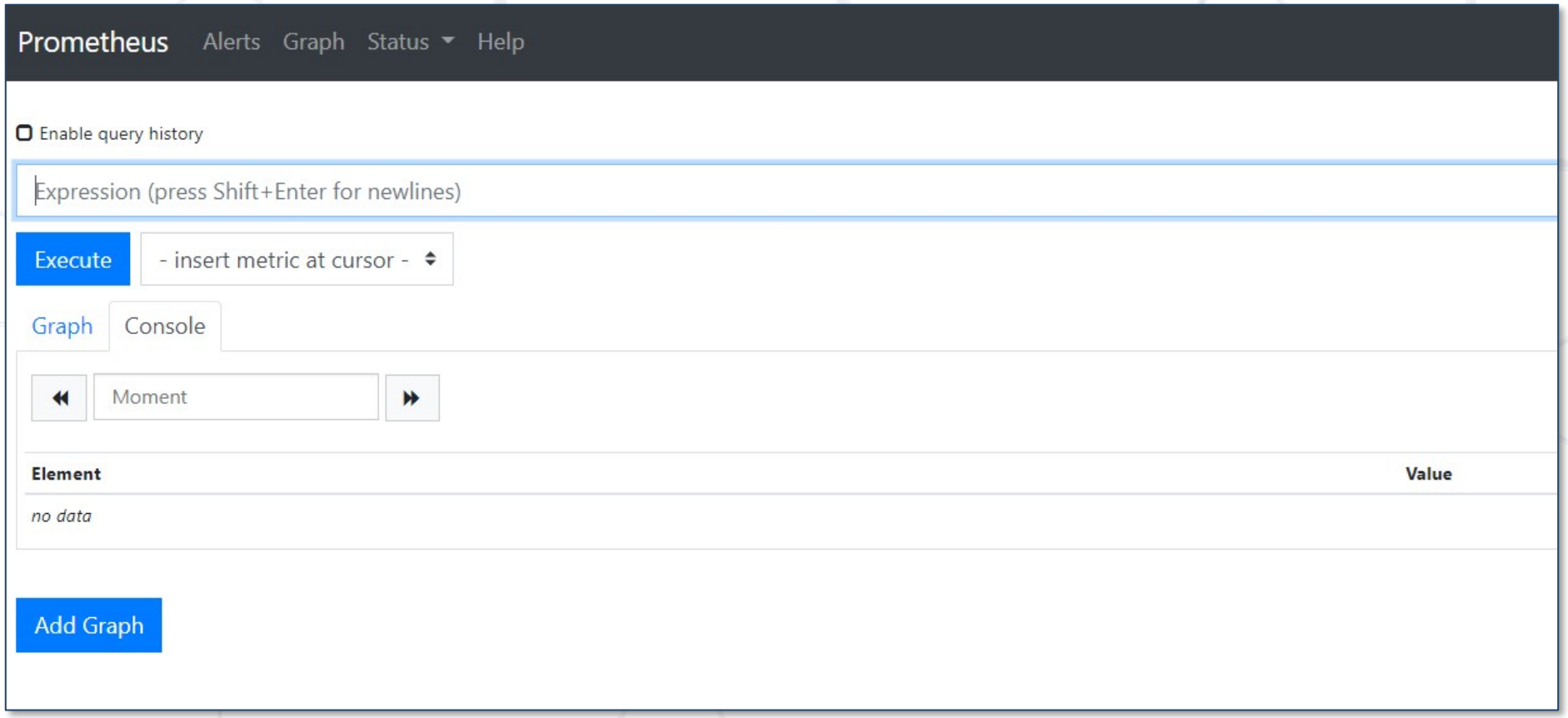
Download and Configure Prometheus

- You can download Prometheus from **here**
- Configure Prometheus with **prometheus.yaml** file

```yaml
global:
  scrape_interval: 15s # By default, scrape targets every 15 seconds.
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from
this config.
  - job_name: 'prometheus'
    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
```

Prometheus Dashboard

- After starting Prometheus, we can access it on http://localhost:9090



Prometheus Query Language – PromQL (1)

- Prometheus provides a functional query language called **PromQL** (Prometheus Query Language)
- Let's the user **select** and **aggregate** time series data in real time
- Result of an expression can either be shown as a **graph**, viewed as **tabular data** in Prometheus' expression browser, or consumed by external systems via the **HTTP API**

Prometheus Query Language – PromQL (2)

- Return all time series with the metric http_requests_total and the **given job** and handler labels

```
http_requests_total{job="apiserver", handler="/api/comments"}
```

- Return a whole **range of time** for the same vector

```
http_requests_total{job="apiserver", handler="/api/comments"}[5m]
```

- Using **regular expressions**

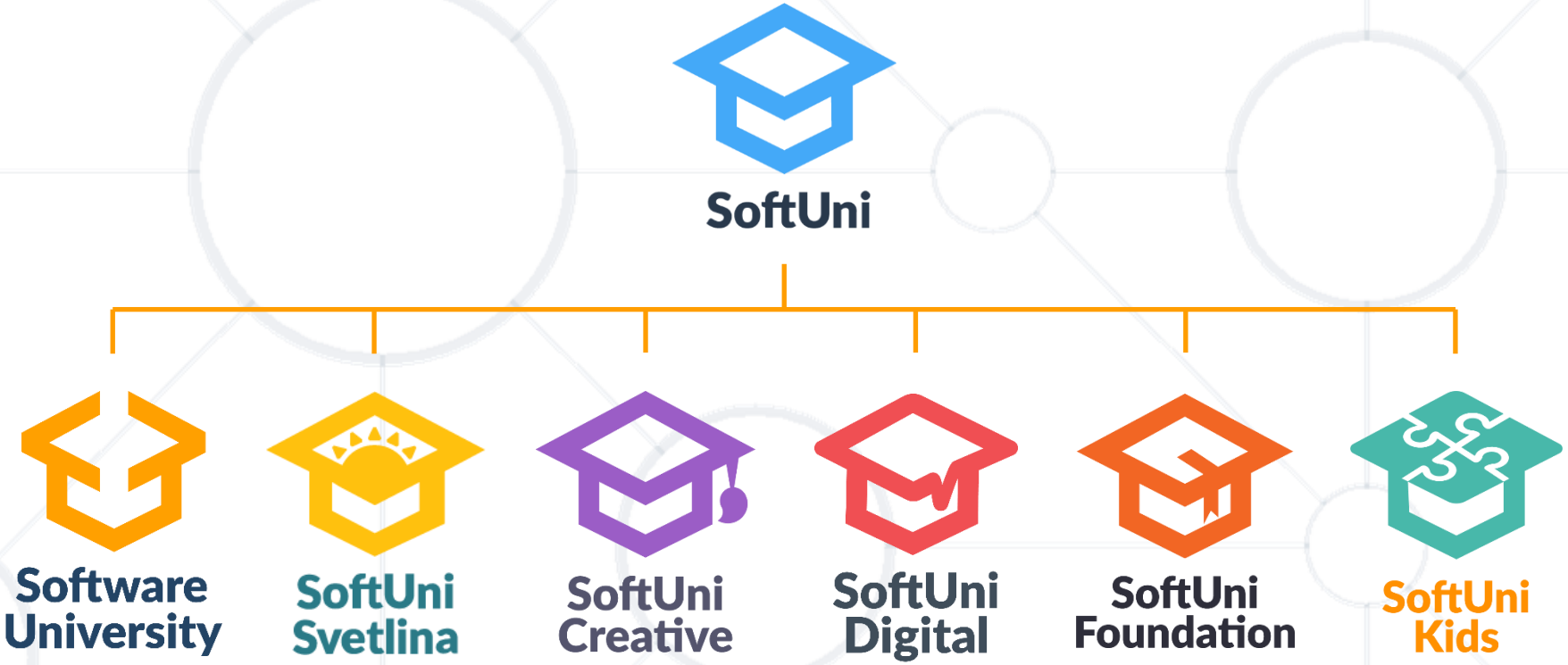
```
http_requests_total{job=~".*server"}
```

```
http_requests_total{status!~"4.."}
```

Summary

- **Deployment** means to push changes or update from one environment to another
- **Micrometer** solves the problem of being a **vendor-neutral data** provider
- **Prometheus** is a Time-series database that **stores the metric data** by pulling it (using a built-in data scraper) **periodically** over HTTP

Questions?



SoftUni Diamond Partners

Educational Partners

VIRTUAL RACING SCHOOL

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg