**Distributed version of the Spring PetClinic Sample Application built with Spring Cloud**

This microservices branch was initially derived from [AngularJS version](https://github.com/spring-petclinic/spring-petclinic-angular1) to demonstrate how to split sample Spring application into [microservices](http://www.martinfowler.com/articles/microservices.html). To achieve that goal we use Spring Cloud Gateway, Spring Cloud Circuit Breaker, Spring Cloud Config, Spring Cloud Sleuth, Resilience4j, Micrometer and the Eureka Service Discovery from the [Spring Cloud Netflix](https://github.com/spring-cloud/spring-cloud-netflix) technology stack.

**Starting services locally without Docker**

Every microservice is a Spring Boot application and can be started locally using IDE ([Lombok](https://projectlombok.org/) plugin has to be set up) or ../mvnw spring-boot:run command. Please note that supporting services (Config and Discovery Server) must be started before any other application (Customers, Vets, Visits and API). Startup of Tracing server, Admin server, Grafana and Prometheus is optional. If everything goes well, you can access the following services at given location:

* Discovery Server - [http://localhost:8761](http://localhost:8761/)
* Config Server - [http://localhost:8888](http://localhost:8888/)
* AngularJS frontend (API Gateway) - [http://localhost:8080](http://localhost:8080/)
* Customers, Vets and Visits Services - random port, check Eureka Dashboard
* Tracing Server (Zipkin) - <http://localhost:9411/zipkin/> (we use [openzipkin](https://github.com/openzipkin/zipkin/tree/master/zipkin-server))
* Admin Server (Spring Boot Admin) - [http://localhost:9090](http://localhost:9090/)
* Grafana Dashboards - [http://localhost:3000](http://localhost:3000/)
* Prometheus - [http://localhost:9091](http://localhost:9091/)

You can tell Config Server to use your local Git repository by using native Spring profile and setting GIT\_REPO environment variable, for example: -Dspring.profiles.active=native -DGIT\_REPO=/projects/spring-petclinic-microservices-config

**Starting services locally with docker-compose**

In order to start entire infrastructure using Docker, you have to build images by executing **./mvnw clean install -P buildDocker**

 from a project root. Once images are ready, you can start them with a single command docker-compose up. Containers startup order is coordinated with [dockerize script](https://github.com/jwilder/dockerize). After starting services it takes a while for API Gateway to be in sync with service registry, so don't be scared of initial Spring Cloud Gateway timeouts. You can track services availability using Eureka dashboard available by default at [http://localhost:8761](http://localhost:8761/).

The master branch uses an Alpine linux with JRE 8 as Docker base. You will find a Java 11 version in the release/java11 branch.

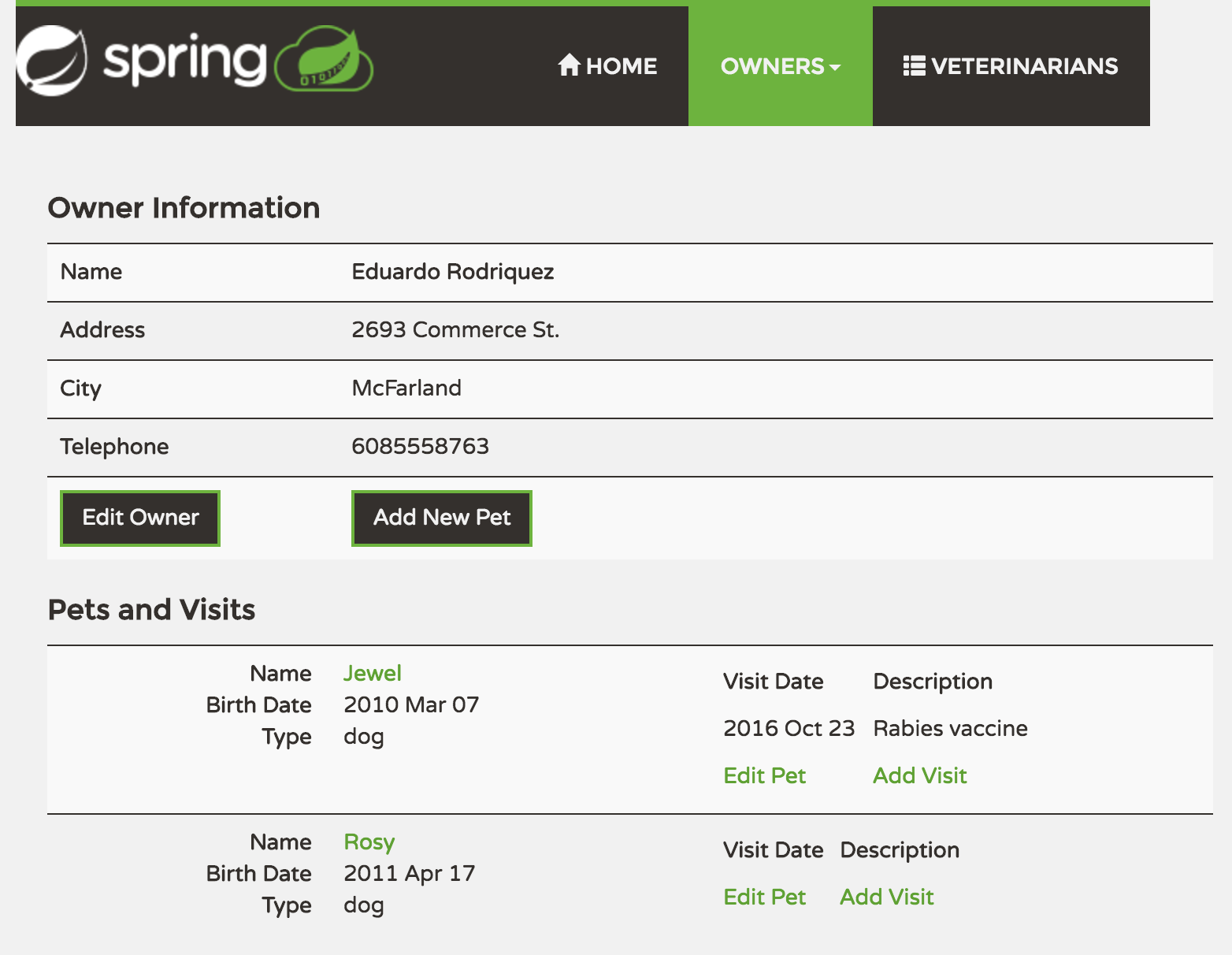
*NOTE: Under MacOSX or Windows, make sure that the Docker VM has enough memory to run the microservices. The default settings are usually not enough and make the docker-compose up painfully slow.*

**Understanding the Spring Petclinic application**

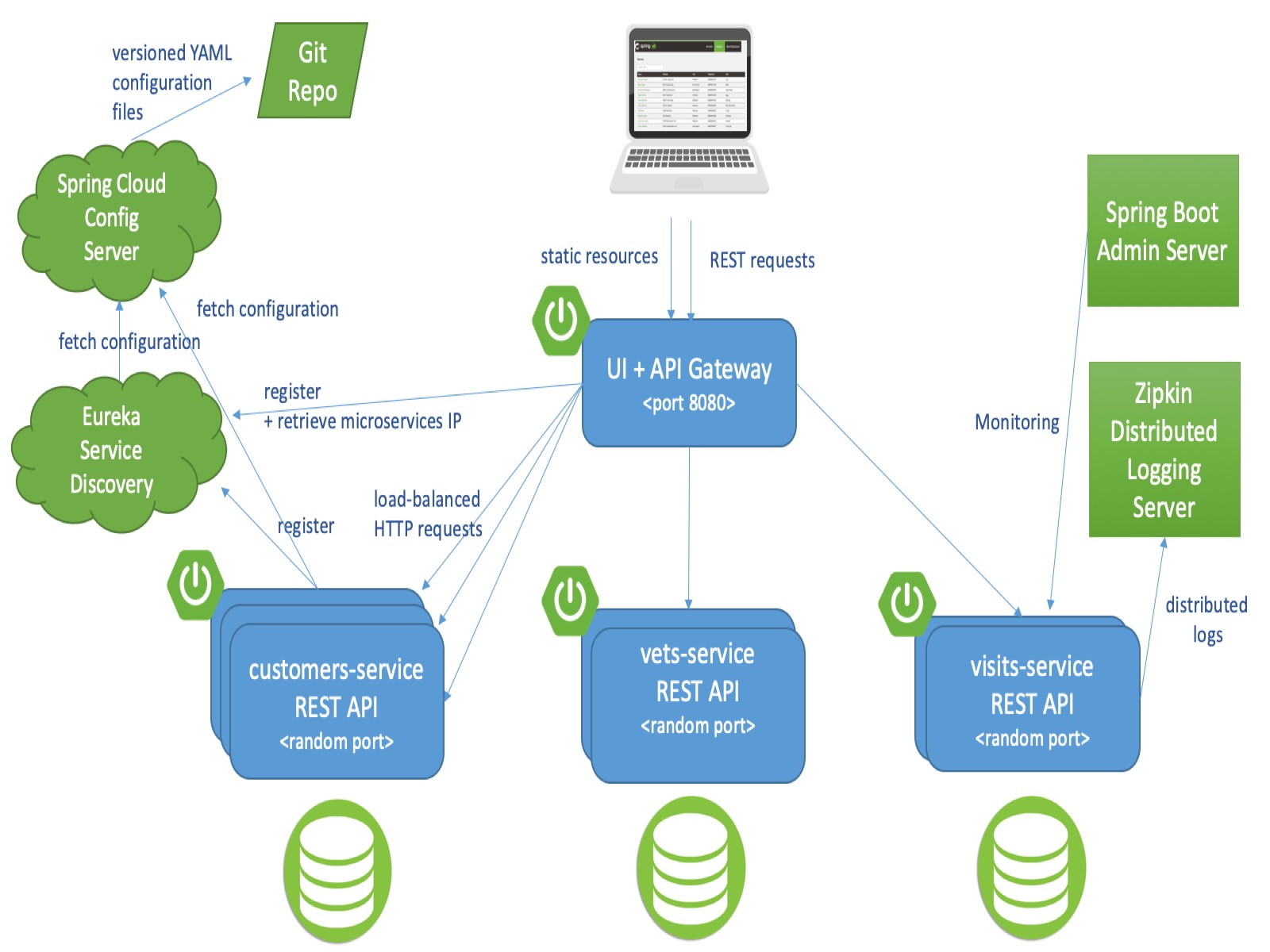
[See the presentation of the Spring Petclinic Framework version](http://fr.slideshare.net/AntoineRey/spring-framework-petclinic-sample-application)

[A blog bost introducing the Spring Petclinic Microsevices](http://javaetmoi.com/2018/10/architecture-microservices-avec-spring-cloud/) (french language)

You can then access petclinic here: <http://localhost:8080/>

[](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/docs/application-screenshot.png)

**Architecture diagram of the Spring Petclinic Microservices**

[](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/docs/microservices-architecture-diagram.jpg)

**In case you find a bug/suggested improvement for Spring Petclinic Microservices**

Our issue tracker is available here: <https://github.com/spring-petclinic/spring-petclinic-microservices/issues>

**Database configuration**

In its default configuration, Petclinic uses an in-memory database (HSQLDB) which gets populated at startup with data. A similar setup is provided for MySql in case a persistent database configuration is needed. Dependency for Connector/J, the MySQL JDBC driver is already included in the pom.xml files.

**Start a MySql database**

You may start a MySql database with docker:

docker run -e MYSQL\_ROOT\_PASSWORD=petclinic -e MYSQL\_DATABASE=petclinic -p 3306:3306 mysql:5.7.8

or download and install the MySQL database (e.g., MySQL Community Server 5.7 GA), which can be found here: <https://dev.mysql.com/downloads/>

**Use the Spring 'mysql' profile**

To use a MySQL database, you have to start 3 microservices (visits-service, customers-service and vets-services) with the mysql Spring profile. Add the --spring.profiles.active=mysql as programm argument.

By default, at startup, database schema will be created and data will be populated. You may also manually create the PetClinic database and data by executing the "db/mysql/{schema,data}.sql" scripts of each 3 microservices. In the application.yml of the [Configuration repository](https://github.com/spring-petclinic/spring-petclinic-microservices-config), set the initialization-mode to never.

If you are running the microservices with Docker, you have to add the mysql profile into the (Dockerfile)[docker/Dockerfile]:

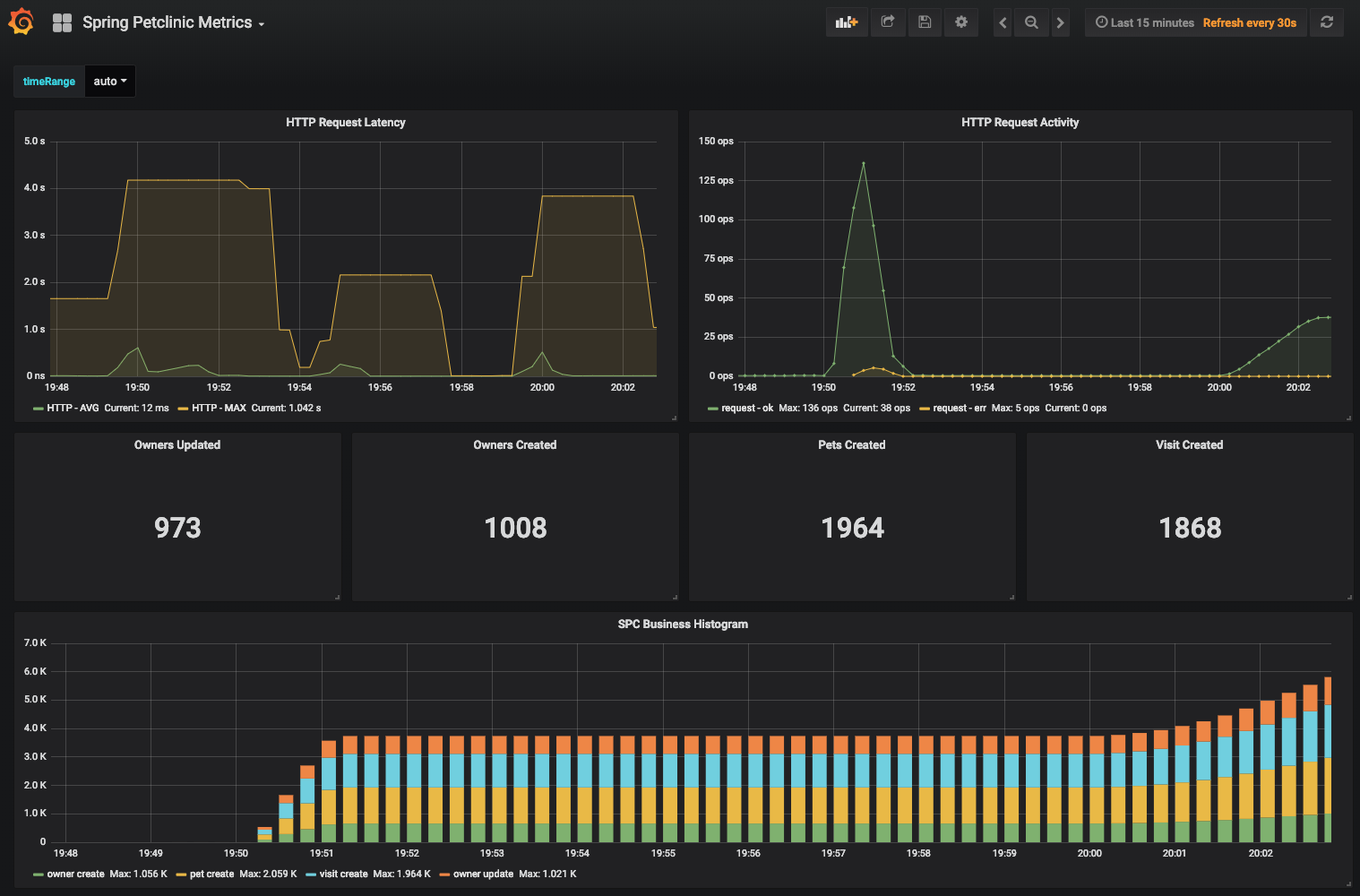
ENV SPRING\_PROFILES\_ACTIVE docker,mysql

In the mysql section of the application.yml from the [Configuration repository](https://github.com/spring-petclinic/spring-petclinic-microservices-config), you have to change the host and port of your MySQL JDBC connection string.

**Custom metrics monitoring**

Grafana and Prometheus are included in the docker-compose.yml configuration, and the public facing applications have been instrumented with [MicroMeter](https://micrometer.io/) to collect JVM and custom business metrics.

A JMeter load testing script is available to stress the application and generate metrics: [petclinic\_test\_plan.jmx](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-api-gateway/src/test/jmeter/petclinic_test_plan.jmx)

[](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/docs/grafana-custom-metrics-dashboard.png)

**Using Prometheus**

* Prometheus can be accessed from your local machine at [http://localhost:9091](http://localhost:9091/)

**Using Grafana with Prometheus**

* An anonymous access and a Prometheus datasource are setup.
* A Spring Petclinic Metrics Dashboard is available at the URL <http://localhost:3000/d/69JXeR0iw/spring-petclinic-metrics>. You will find the JSON configuration file here: [docker/grafana/dashboards/grafana-petclinic-dashboard.json](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master).
* You may create your own dashboard or import the [Micrometer/SpringBoot dashboard](https://grafana.com/dashboards/4701) via the Import Dashboard menu item. The id for this dashboard is 4701.

**Custom metrics**

Spring Boot registers a lot number of core metrics: JVM, CPU, Tomcat, Logback... The Spring Boot auto-configuration enables the instrumentation of requests handled by Spring MVC. All those three REST controllers OwnerResource, PetResource and VisitResource have been instrumented by the @Timed Micrometer annotation at class level.

* customers-service application has the following custom metrics enabled:
  + @Timed: petclinic.owner
  + @Timed: petclinic.pet
* visits-service application has the following custom metrics enabled:
  + @Timed: petclinic.visit

**Looking for something in particular?**

| **Spring Cloud components** | **Resources** |
| --- | --- |
| Configuration server | [Config server properties](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-config-server/src/main/resources/application.yml) and [Configuration repository](https://github.com/spring-petclinic/spring-petclinic-microservices-config) |
| Service Discovery | [Eureka server](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-discovery-server) and [Service discovery client](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-vets-service/src/main/java/org/springframework/samples/petclinic/vets/VetsServiceApplication.java) |
| API Gateway | [Spring Cloud Gateway starter](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-api-gateway/pom.xml) and [Routing configuration](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-api-gateway/src/main/resources/application.yml) |
| Docker Compose | [Spring Boot with Docker guide](https://spring.io/guides/gs/spring-boot-docker/) and [docker-compose file](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/docker-compose.yml) |
| Circuit Breaker | [Resilience4j fallback method](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-api-gateway/src/main/java/org/springframework/samples/petclinic/api/boundary/web/ApiGatewayController.java) |
| Grafana / Prometheus Monitoring | [Micrometer implementation](https://micrometer.io/), [Spring Boot Actuator Production Ready Metrics](https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-metrics.html) |

| **Front-end module** | **Files** |
| --- | --- |
| Node and NPM | [The frontend-maven-plugin plugin downloads/installs Node and NPM locally then runs Bower and Gulp](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-ui/pom.xml) |
| Bower | [JavaScript libraries are defined by the manifest file bower.json](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-ui/bower.json) |
| Gulp | [Tasks automated by Gulp: minify CSS and JS, generate CSS from LESS, copy other static resources](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-ui/gulpfile.js) |
| Angular JS | [app.js, controllers and templates](https://github.com/spring-petclinic/spring-petclinic-microservices/blob/master/spring-petclinic-ui/src/scripts) |

**Interesting Spring Petclinic forks**

The Spring Petclinic main branch in the main [spring-projects](https://github.com/spring-projects/spring-petclinic) GitHub org is the "canonical" implementation, currently based on Spring Boot and Thymeleaf.

This [spring-petclinic-microservices](https://github.com/spring-petclinic/spring-petclinic-microservices/) project is one of the [several forks](https://spring-petclinic.github.io/docs/forks.html) hosted in a special GitHub org: [spring-petclinic](https://github.com/spring-petclinic). If you have a special interest in a different technology stack that could be used to implement the Pet Clinic then please join the community there.