UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO

Dipartimento di Scienze di Base e Fondamenti

Scuola di Scienze e Tecnologie dell'Informazione

Tesi di Laurea

DEVELOPMENT, TESTING, AND CONTINUOUS INTEGRATION OF CONTAINERIZED WEB APPLICATIONS

Relatore:
Chiar.mo Prof. Alessandro Bogliolo

Candidato:
Michele Sorcinelli

Correlatori:
Dott. Luca Ferroni
Dott. Andrea Seraghiti
Dott. Lorenz Cuno Klopfenstein

Corso di Laurea in Informatica Applicata

Anno Accademico 2014-2015

Ai miei genitori

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work aims to cover the whole development process of modern web applications organized in lightweight environments called *containers*. It shows how development can be made easier with containers adoption, and how automated testing with a good test suite, together with a Continuous Integration system, effectively improve the software quality because they help in discovering bugs and act by themselves as quality indicators.

A container is an isolated environment where an application resides and runs. Containers can communicate to each other using a virtual network provided by the *container engine*, which is also the software that creates and manage them. A container based virtualization approach leads to different advantages in production, in terms of scalability, reliability, automation, better management, and rolling updates.

In the development of web applications, containers can be used to obtain a development environment similar to the production environment, eliminating the disadvantages arising from the differences between these two environments, such as bugs related to different versions of one or more application components. Furthermore, providing a containerized development environment helps the reproduction of reported bugs, given that all the developers work in a standardized, reproducible, and easy to obtain environment.

Part of the research work preceding the thesis consisted in comparing different typologies of tests that could be implemented for a software, and finding the most suitable and urgent for the case of study taken into account. The testing typology chosen for this thesis work is *end-to-end*. End-to-end testing is the testing of the final product, and its purpose is to simulate in a programmatic way the user interaction with the application interface, in order to discover bugs that are visible from the external. Those bugs are the same that the users discover first, and thus the most urgent to deal with. End-to-end testing for web applications involves systems to drive different web browsers, make them interacting with the application and so ensuring that the application features are working properly.

Regarding the testing of the software, there is a series of steps to perform in order to adapt the testing process to the container based virtualization, gaining the advantages of a standardized testing environment that is completely reproducible. Often developers don't run integration and end-to-end tests on their desktop or laptop machines because they don't want to configure the testing environment, or different configurations can lead to different test results. This work will cover the aspects of integrating the container approach to the software testing, and will provide simple procedures to run all the implemented tests without effort for the developers.

A *SCM* (Source Code Management) system is a software used to manage the source code of one or more projects, registering every change and addition introduced from its born to the last version. A Continuous Integration system is a software that provides automatic check of software projects, building and testing the applications at every revision. Tests can run in the developers machines, but it is mostly important that they run also in a Continuous Integration system. Once revision are uploaded in a SCM system, automated tests have to be run (Continuous Integration), and if the tests pass, then the application can be automatically deployed (Continuous Delivery), else, the deploy must abort and developers have to be warned about the critical situation.

This work is the outcome of a collaboration with the *beFair* software development team, that begun at the firsts months of 2015, when the team realized that a boost in the quality of the development process and final product could be obtained with the application of automated software testing, and, at the same time, it decided to change the way applications are developed and deployed, shifting from a traditional development and deploying approach, to a containerized one.

Gasista Felice [3], developed by the beFair team and released as *FLOSS* (Free Libre and Open Source Software) [13], is the use case for this work. Gasista Felice is an online management application made for GAS (Ethical Purchasing Groups) initially developed for *DES* (Solidarity-based Economy District) Macerata.

1.1 Organization

Chapter 2 exposes the technological and social context that led to the thesis work, as well as the motivation of choosing the end-to-end testing typology and the container based virtualization.

Chapter 3 introduces and explains in detail the *Docker* [5] container engine, with practical examples of usage, then it shows how the *Docker Compose* [8] container orchestration tool can be used to simply define and build portable development and testing environments. The last section of the chapter exposes the structure of Gasista Felice, as well as its configuration of the development environment.

Chapter 4 covers the implementation of browser automation routines with the *Protractor* [2] framework for the end-to-end testing of *AngularJS* [15] web applications, and in particular, of Gasista Felice AngularJS web client.

Chapter 5 shows the main tools used in end-to-end tests and how to configure the testing framework to work with the Gasista Felice application container ecosystem. The configuration is an integral part of this thesis and aims to reduce the developer effort to get the testing environment.

Chapter 6 explains how to build a Continuous Integration system to provide automatic run of tests once the code is updated and pushed into the project repository. Continuous Integration is essential for Continuous Delivery and rolling release of the software products.

Chapter 2

Context

Computer Science is a young discipline, but even if young it drastically evolved over a few decades. The diffusion of personal computers and the internet, as well as mobile devices and cloud computing in the last years led to an exponential increase of information technology related products and services usage, thus incrementing also the necessity of productivity improvement in software development organizations.

To overcome this requirement, different approaches have been implemented in different areas of interest in software development (Table 2.1), in particular in terms of new software architectures, changes in infrastructure and in the workflow adopted during development.

Table 2.1: The evolution of Information Technology

	1990's	2000's	2010's
Architecture	Monolith	Layered Monolith	Microservices
Infrastructure	Phisical Machines	Virtual Machines	Containers
Dev. Workflow	Waterfall	Agile	DevOps

This chapter provides an overview of how Information Technology evolved in these categories and how both this thesis project and the *Gasista Felice* web application represent the current state of the art.

2.1 Evolution of software architectures

The first developed applications in the history of information technology weren't equipped with a well defined software architecture. For simple, small, applications, developed by few people (if not single developers), the absence of a defined software architecture wasn't a problem, but in bigger projects this absence often led to examples of so called *spaghetti code* (code with a complicated structure, difficult to read and maintain), that brought entire projects to failure. The situation improved when developers began to divide the code using a modular approach, as suggested by the Unix philosophy [12], assuring reuse and a better maintenance of the projects.

2.1.1 Model View Controller

Regarding web applications, a common adopted software architecture is the *MVC* (Model View Controller), that consists in the separation of the application in three distinct components:

- A model for the storage of the application data
- A view for displaying a representation of the data contained in the model
- A controller that acts as intermediary between the model and view, manipulating the model

A web application that adopts this pattern permits a good separation of concerns, but is still a layered monolith application, and even if a layered monolith application is divided in modules, often the code is not well decoupled, thus leading to circular dependencies among them and thus rending changes to a module more difficult and frustrating. A monolith application can grow in a unexpected way, making the project hard (if not impossible) to maintain.

2.1.2 From MVC to Microservices

> *The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services.* [25]

In the microservices software architecture, the project is developed as a set of small and lightweight services, that communicate amongst them through a well defined API. These small services can be written in different languages, and replaced when they are obsolete.

In addition of deployment advantages (such as scalability) of microservices structured applications, in software development a microservices approach can be seen as a new implementation of the Unix principles of modularity: the services themselves acts as modules for a software, and are completely independent.

In this thesis project the Gasista Felice web application has been containerized, so its infrastructure and development process has been prepared to embrace a migration to the microservices software architecture. In fact, the container infrastructure is the key of development and deployment of microservices structured applications.

2.2 Software infrastructures and development environments

In the past web applications were deployed in physical machines, leading to different problems in management and maintenance due to their physical and so *static* constraints. Virtual machines permitted to resolve these problems, delegating the management of the physical server machines to third parties, thus freeing the organization from this duty.

While virtual machines emulate every aspect of the desired machine and operating system, those features are heavy and often unnecessary, and that's the reason of the diffusion of containers. Containers are lighter than virtual machines, because they don't aim to reproduce an entire machine, but only the environment required to run a particular software. The Docker container engine use the virtualization features of the Linux kernel in order to reproduce a software operating environment in a native and lightweight way. Thanks to their lightweight nature, containers seems today to be the perfect infrastructure for the development and deployment of applications with a microservices architecture.

In the development of a web application with a traditional infrastructure, developers install all the required tools, such the database management system or the application framework, in their machines, through the installation procedures available for their operating system. These installation procedures are complicated and frustrating, and distract the developers from the development activity. In addition to that, the development environment created trough the installation and configuration performed by a developer can be different to the environment created by another developer due to the differences between their operating systems, and these differences can lead to difficulties when attempting to reproduce and fix a reported bug that is related to a particular environment.

These problems led to the diffusion of virtualization tools, that permits to formally define the development environment and his features, and to virtually reproduce them in every machine. Virtualization can be obtained both with virtual machines and containers.

After the containerization of the Gasista Felice application, this thesis work will show how this change affects the whole development environment.

2.3 Development Workflow

A classic development workflow is the *Waterfall* model, that consists in the following steps (that can be adapted):

1. Requirements specifications
2. Design

3. Implementation
4. Verification
5. Maintenance

Given that the steps are executed sequentially, this model provides a very slow development, integration and delivery process.

2.3.1 Agile and Continuous Integration

A more modern model is provided in the *Manifesto for Agile Software Development*:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.* [20]

In the Agile model the highest priority is the customer satisfaction, reached trough collaboration, and early and continuous delivery of the software [21]. In order to actually provide continuous delivery and flexibility to changes, Continuous Integration is essential.

Continuous Integration is a practice where the members of the development team integrates their work frequently (daily or even more frequently). Every integration is verified by an automated system that download the last software revisions, and run automated tests to verify the correctness of the application in its entirety [24].

This thesis work will cover the implementation of automated tests for the Gasista Felice project and the installation and configuration of a Continuous Integration system, as well as its integration with the container infrastructure of Gasista Felice and with a SCM (Source Code Managament) system for the automatic triggering of builds.

DevOps

DevOps [26] (Development and Operations) is an extension of the Agile workflow that adds the priority of establishing a parity between development and production environments and promotes collaborations between developers and IT operators.

In this context Continuous Integration is associated with the Continuous Delivery of the software product. Continuous Delivery consists in automatic deploy of the software as soon as the Continuous Integration system verified its correctness.

The necessity of a Continuous Delivery system is covered by this work with the installation and configuration of the Continuous Integration system.

2.4 Testing

There are different categories of automated tests, and in particular:

- Unit tests
- Integration tests
- End-to-end tests

Unit tests regard the testing of computational units. A computational unit can be seen as the smallest testable part of an application (for example a class or a method in OOP), and is performed by executing the unit with an input parameter and evaluating the returned results, that has to be the expected. The value expected as result should be hardcoded (the first error made by unexperienced testers is to think that testing routines should do computation, while testing routines have the role of calling program procedures with a set of fixed parameters and comparing the return value with the expected).

Sometimes the called methods needs the interaction with an external component, such as a database; in that case, the interaction with the component has to be replaced with the interaction with a dummy object that usually has the role of returning a fixed value.

Integration tests regard the testing of a grouped set of modules in order to verify their interaction. It can involve the interaction with external components, and in that case, the testing environment has to be configured in order to permit this interaction (for example an integration test that involve a database query has to be executed in an environment provided with a test database).

End-to-end tests, sometimes called system tests, regard the testing of application from the user point of view. For web application end-to-end are implemented with browser automations scripts, that require the configuration of a software that act as a driver for different browsers in order to make them access the web application and test its features.

The chosen typology for starting the implementation of automated tests in the Gasista Felice application is the last one, for different reasons:

- End-to-end tests operate in the outer layer of the application, thus have an higher probability of discovering bugs because they also trigger the executions of procedures in all the application underlying layers
- End-to-end tests can be used in order to ensure that the application requirements are correctly satisfied, and can be used in the interactions with the customer as demonstration of the implemented features
- Bugs discovered by end-to-end tests are the most evident for the customer, thus the most urgent
- Configuring the container environment to include containers for the testing framework will permit the running of end-to-end tests without additional configuration to be done by developers in the future, while unit tests can also be executed without additional configurations

A particular importance is attributed to the last point, because after the containers configuration is done, developers can implement and run both unit and end-to-end tests, opening the way for the Test Driven Development, the practice of writing tests as specifications of a feature to implement, before actually implement that feature. Before writing a new method, the developer can write a unit test to verify its behaviour, and before implementing a feature requested by the customer, a end-to-end tests can be written as specification of that feature. With this approach, the last implemented features are also covered with automated testing, drastically improving the quality of the software.

2.5 Gasista Felice

Gasista Felice [3] is an online management application made for *GAS* (Ethical Purchasing Groups) by the beFair team, and initially developed for *DES* (Solidarity-based Economy District) Macerata.

Gasista Felice is the use case for the thesis work, and in particular, is infrastructure, that will be containerized, and its *AngularJS* based web interface, that acts as use case of end-to-end tests implementation.

The beFair team is involved in *Sharing Economy* [4], that focuses on a major interaction between providers and consumers, enhances the community, creates discussion places, and provides direct feedback.

2.5.1 Free Libre and Open Source Software

This work is made, in its entirety, with FLOSS (Free Libre and Open Source Software), and is also released as FLOSS.

The beFair team is oriented to FLOSS principles, that are explained in detail in by the *GNU* (GNU is Not Unix) project website:

> *"Free software" means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, "free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer". We sometimes call it "libre software" to show we do not mean it is gratis.* [13]

In fact, free doesn't mean gratis, but means that if the user obtains a copy of the software, obtain also the right to do all the things listed above.

Free software ensures communication and collaboration with the customer, transparency, reuse, better feedback and lower release times, so it's an added value to a software product, and fits well with the Sharing Economy principles.

Chapter 3

Containers and containerized applications

In the past virtual machines covered a role of great importance in cloud computing and virtualization, because of their attitude to provide standardized and reproducible environments, installable everywhere. The main problem with Virtual Machines is that their configuration concerns also with the detail of the machine, such as hardware (RAM, hard disk, processor, network interfaces, etc.), when often those features don't require reproduction and introduce overhead. In fact, the most important part of the virtualization is the operating system and application configuration, features that are reproduced with fidelity by application containers. In addition, the only processes running inside a container are the application ones, while inside a virtual machine the processes of an entire operating system run, increasing significantly the overhead.

The elimination of the overhead and complexity introduced by virtual machines is not the only reason to prefer containers: *Docker* container engine provides versioning of the images used for container creation, that is a great benefit for software development, in fact versioning practices are adopted in all the software development teams and software houses. Also Docker provides component reuse: a base image can be reused by an infinite number of applications, reducing impact on disk space and build times. Starting from version 1.8, released on August 11, 2015, Docker provides image signing and verification for improved security. [9]

This chapter exposes the main features of *Docker* container engine and *Docker Compose*, a container orchestration tool for defining and instancing reproducible development environments. Once the reader is familiar with the main features of Docker and Docker Compose, the containerized structure of Gasista Felice is explained in detail.

3.1 Docker container engine

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux, Mac OS and Windows. [28]

Docker permits to build images of applications that can be instanced as containers. For any container, Docker provides an isolated and reproducible environment with the advantage of accessing directly to the virtualization features of the Linux kernel (Figure 3.2), avoiding the overhead of installing and maintaining virtual machines (Figure 3.1). To access the virtualization features, Docker can use different interfaces such as *libcontainer*, *libvirt*, *LXC* (Linux Containers) and *systemd-nspawn* [28]. Docker is written in the *Go* programming language.

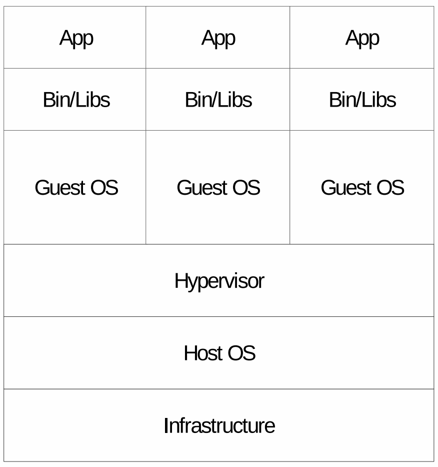


Figure 3.1: Applications running in virtual machines

3.1.1 Installation

Docker can be installed in Debian with:

```
# apt-get install docker.io
```

Instructions for other GNU/Linux distributions can be found on Docker online documentation. For Windows and Mac OS X, on August 11, 2015, *Docker Toolbox* has been released as new installer replacing the older *Boot2Docker*.

The Docker Toolbox is an installer to quickly and easily install and setup a Docker environment on your computer. Available for both

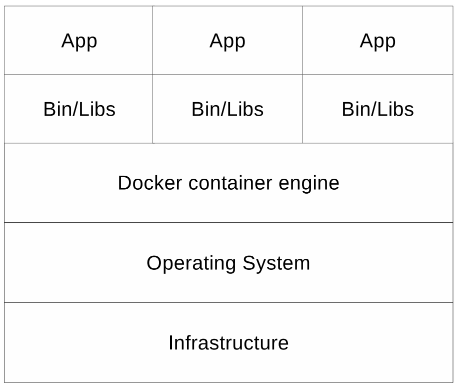


Figure 3.2: Applications running in Docker containers

> *Windows and Mac, the Toolbox installs Docker Client, Machine, Compose (Mac only), Kitematic and VirtualBox. [10].*

It's obvious that the only operating systems that can run Docker natively (without the support of a virtual machine) are those that run on the Linux kernel.

3.1.2 Docker images

The base for creating a Docker container is an image of an application. The main repository of Docker images is *Docker Hub* [7], where images for all the most famous open-source applications can be found. Any user can sign in to Docker Hub and push an image, or make it build on the server. In order to create an image, a *Dockerfile* with the specification of the environment has to be written.

For example, this thesis is built using *LaTeX*, that is a powerful language for document generation. Instead of installing LaTeX and all the required packages, a prebuilt and ready to use image can be used to process LaTeX source code. With Docker, compile this thesis in every supported operating system [6] is just a matter of running this command on a shell from the thesis directory:

```
$ docker run -v $PWD:/code michelesr/latex pdflatex thesis.tex
```

This command will search for the `michelesr/latex` image in the system, and if is not found, will pull the image from Docker Hub. Then the working directory, that contains the LaTeX source files, will be mounted to `/code/` inside the container, and will be used as working directory. At last, the command will run and the thesis will be magically compiled even if LaTeX is not installed in the system. The Dockerfile used to build the LaTeX image is the following:

```
FROM debian:8

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

ENV DEBIAN_FRONTEND         noninteractive

RUN apt update && \
    apt install -y texlive-full && \
    rm -rf /var/lib/cache/apt/* \
           /var/lib/apt/lists/*

RUN adduser latex --shell /bin/bash
RUN mkdir /code/ && chown latex: -R /code/

USER latex
WORKDIR /code/
```

Starting from a *Debian* 8 image, it adds the environment variable `DEBIAN_FRONTEND` and sets it to `nointeractive` to tell Debian that a non interactive script is running, then runs `apt` to install the required packages and `rm` to remove useless files from apt cache and lists. Once the required packages are installed, it creates the `latex` user and the `/code/` directory in the filesystem root, setting `latex` as its owner. Finally it sets `latex` as the default user and `/code/` as the default working directory.

The reason behind the running of the `latex` command as `latex` user is correlated to the ownership of the files generated from LaTeX processing. Inside the environment, the `latex` user has `1000` as UID (User Identifier) and GID (Group Identifier), that are usually the default values for a desktop user in Unix-like operating systems. This trick made the generated files directly accessible from the host system user after the compilation. If for some reason different UID or GID are required, they can be set in the Dockerfile modifying the `adduser` instruction:

```
RUN adduser latex --uid UID --gid GID --shell /bin/bash
```

Once a Dockerfile for the image has been defined, it can be reused in every operating system running Docker, with the guarantee that the produced effects are always the same (with the exception of kernel dependent behaviours).

3.1.3 A layered approach

The Dockerfile for the previous image was built using `debian:8` image as base. Instead of distributing an image as standalone, Docker use a smart layered approach: for every instruction of the Dockerfile, it adds a layer to the base image, then these layer are cataloged using a hash algorithm. The final image is built overlapping all the layers, allowing the reuse of these to build other images if necessary. For example, if the above Dockerfile user is modified, only the layers after their modification are rebuilt. Also if another image that use `debian:8` as base is built or pulled, the `debian:8` layers are reused. This approach provides a significant boost of build speed and reduction of disk usage.

3.1.4 Running containers as daemons

Containers can be also used to run daemon applications, such as web servers. For example, to run *Gogs*, a *Git* [1] Service Application for software versioning written in *Go*, the command is:

```
$ mkdir /var/gogs
$ docker run -d -p 3000:3000 -v /var/gogs:/data gogs/gogs
```

As for the LaTeX image, a volume containing the application configuration files is mounted, but also the `-d` parameter is used to inform Docker that the software has to be launched as daemon, and `-p 3000:3000` is used to expose the service outside the container. Once the daemon is up, the web application can be visited at `http://localhost:3000`. The application log can be inspected:

```
$ docker logs -f container_id
```

Container name or hash can be used as ID, and the `-f` parameter allows the continuous prompt of new log entries.

3.1.5 Running a shell inside a container

Sometimes is useful to run a shell inside the environment of a container. In order to launch a bash shell inside a container the command is:

```
$ docker run --rm -it image_id /bin/bash
```

Image name or hash can be used as ID. The `-i` stands for `--interactive`, and the `-t` is used to allocate a virtual terminal device for the shell. The `--rm` option is used to destroy the container after the exit of the shell. The destruction can be safely performed because once a container is started (for example as a daemon) with a command, will continue to execute that command until its death. Containers can be stopped, started and restarted, but once a container is created, the command cannot be changed.

Even if the container is isolated and launching a command inside it can seems meaningless, is useful when containers are linked in network. For example a shell running inside a container that is linked to a database container can be used to perform data manipulation or dump.

3.1.6 Containers and images management

The default behaviour of Docker is to leave the stopped containers saved to allow their restart, unless the `--rm` option is used. The running containers can be listed with:

```
$ docker ps
```

To list all the containers, including stopped ones, the `-a` parameter can be used. To remove a stopped container:

```
$ docker rm container_id [container2_id ...]
```

Container names or hashes can be used as ID. With the `-f` parameter, the remotion of running containers can be forced. To remove all the stopped containers in the system:

```
$ docker rm $(docker ps -aq)
```

The `-q` parameter in the inner command lists the hashes of all the containers in the system, then their hashes are used as container ID for their remotion (running containers are ignored unless `-f` is provided). To lists the installed images:

```
$ docker images
```

To remove images:

```
$ docker rmi image_id [image2_id ...]
```

As for the container, ID is the image name or hash. The `-f` option can be used to remove an image even if containers for that image are instanced. The directory used for storing of Docker data is `/var/lib/docker/`, and it contains:

- images
- containers
- metadata of images and containers
- temporary files

In particular, the `/var/lib/docker` directory can grow unexpectedly, completely filling the machine disk. To avoid the exhaustion of the disk space, a periodic cleaning of unused images, containers, and of the `/var/lib/docker/tmp` directory has to be performed.

Image names and untagged images

Due to the layered approach used by Docker, an image is composed of more image layers overlapped. When the `-a` parameter is added to `docker images`, even the untagged images are shown. Untagged images haven't a name but are used as layer of other images. Tagged images have the form of `repository:tag`, where:

- `repository` is the name of the image repository, found in the form `author/name` or `name` for official images (such as `debian`)
- `tag` distinguish between differents version of the image, and usually consist in a version number (`iojs:3`) or version name (`debian:jessie`) or `latest` for the latest version available (`michelesr/latex:latest`)

When tag is not specified during image building or pulling, `latest` is used. Sometimes the same image is referred with more tags:

```
$ docker images | grep iojs
iojs   3         becb6285c124  5 days ago  634.8 MB
iojs   latest    becb6285c124  5 days ago  634.8 MB
```

When a remotion of a image is requested with the `docker rmi` command and a image name is provided as ID, Docker untags the image, removing the reference:

```
$ docker rmi iojs:3
Untagged: iojs:3

$ docker images | grep iojs
iojs   latest    becb6285c124  5 days ago  634.8 MB
```

Then if all the references for the image are removed and there aren't other images that depends on it, the image is deleted from disk:

```
$ docker rmi iojs:latest
Untagged: iojs:latest
Deleted: becb6285c1246f732efe6e90ac8931acab01be09031d97a4fc60e1b0b357309d
Deleted: 2a153eb979fa93f1601fc89ab8ae301db02d7239d32286fc629e38a629c407b2
Deleted: 79e9c6d779863a9df07ca0c5b59b18acc7d9e4c955547f37d5738e22cb63cbe7
```

Also the `2a153eb979fa` and `79e9c6d77986` images are deleted because they were used only as layer for the `becb6285c124` image. An image that is used as a base for other images cannot be removed, for example trying to remove the `debian:8` image lead to the following result:

```
$ docker images | grep debian
debian    8    9a61b6b1315e    4 weeks ago    125.2 MB

$ docker rmi -f debian:8
Untagged: debian:8

$ docker images -a | grep 9a61b6b1315e
<none>    <none>    9a61b6b1315e    4 weeks ago    125.2 MB

$ docker rmi -f 9a61b6b1315e
Error response from daemon: Conflict, 9a61b6b1315e wasn't deleted
Error: failed to remove images: [9a61b6b1315e]
```

The `9a61b6b1315e` cannot be removed because one or more images depends on it (for example the `michelesr/latex` is built using `FROM debian:8`). The `debian:8` name and tag can be reassigned:

```
$ docker tag 9a61b6b1315e debian:8

$ docker images | grep debian
debian    8    9a61b6b1315e    4 weeks ago    125.2 MB
```

3.1.7 Containers linking

Docker use a VPN to allow communication between containers. To exploit this feature, the `--link` parameter can be used to create a container that is linked with one or more containers. For example:

```
$ docker run -d --name foobar nginx
d3dd23d4b9c688267ad04378a4cc8d71674732e73c5b6b388e7ff740e767c7af

$ docker run --rm -it --link foobar:webserver debian /bin/bash
root@59633390aff6:/# ping webserver -c 2
PING webserver (172.17.1.210): 56 data bytes
64 bytes from 172.17.1.210: icmp_seq=0 ttl=64 time=0.249 ms
64 bytes from 172.17.1.210: icmp_seq=1 ttl=64 time=0.164 ms
--- webserver ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.164/0.207/0.249/0.043 ms
```

In the example above, `foobar` has been used as `--name` for the `nginx` container that has been launched as a daemon. At the launch of the Debian container, Docker links the container named `foobar`, registering an entry in `/etc/hosts` with the hostname `webserver` referring to the `foobar` container.

```
root@59633390aff6:/# cat /etc/hosts | grep webserver
172.17.1.210    webserver d3dd23d4b9c6 foobar
```

Containers linking is the base of serving containerized web applications, that usally are divided in different containers, such as:

- Database
- Application Server
- Proxy Server
- Web Interface
- Testing containers

To simplify the process of building, linking and managing a containerized application, *Docker Compose* can be used.

3.2 Docker Compose

Distributed applications consist of many small applications that work together. Docker transforms these applications into individual containers that are linked together. Instead of having to build, run and manage each individual container, Docker Compose allows you to define your multi-container application with all of its dependencies in a single file, then spin your application up in a single command. Your application's structure and configuration are held in a single place, which makes spinning up applications simple and repeatable everywhere [8].

Docker Compose is a powerful tool for the development of containerized applications. With Docker Compose, the entire application structure can be defined in a single configuration file called `docker-compose.yml`, and instanced with a singe command. Docker compose is written in the *Python* programming language.

3.2.1 Installation

Docker Compose can be installed with *PIP*, the Python package manager:

```
# pip2 install docker-compose
```

Python 3 is not supported. Docker Compose is only a wrapper to Docker functions, and Docker has to be installed in the system. PIP can be installed in Debian with the command:

```
# apt-get install python-pip
```

Currently Docker Compose is not supported on Windows operating system. Install instructions for other GNU/Linux distributions can be found on Docker Compose online documentation. Docker Toolbox for Mac OS X includes Docker Compose.

3.2.2 The docker-compose.yml configuration file

The `docker-compose.yml` configuration file Docker Compose contains a declarative description of the application containers to instantiate, link and run. The syntax used by this configuration file is *YAML*, a language for data serialization (like *JSON*).

An example of a simple web app configuration consists in a Dockerfile for the application server and a `docker-compose.yml`:

```
# ./Dockerfile

WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code
CMD python app.py

# ./docker-compose.yml

web:
  build: .
  volumes:
  - ./data:/data
  links:
  - db
  ports:
  - "8000:8000"
db:
  image: postgres
```

This configuration is used to build an application that consists of two containers:

- `web`: a Python web application server
- `db`: the *PostgreSQL* database management system

Docker Compose use a simple syntax to define ports exposing, volumes mounting, and containers linking. All these functions are wrapped from Docker container engine, so they work exactly as explained previously. In this example the `web` component image is built from the Dockerfile, while the image for `db` is pulled from Docker Hub image registry. To build and run the application:

```
$ docker-compose up -d
```

The `-d` parameter is provided to detach the application process from the shell in order to launch the application as daemon. The complete list of Docker Compose functions is:

```
build              Build or rebuild services
help               Get help on a command
kill               Kill containers
logs               View output from containers
port               Print the public port for a port binding
ps                 List containers
pull               Pulls service images
restart            Restart services
rm                 Remove stopped containers
run                Run a one-off command
scale              Set number of containers for a service
start              Start services
stop               Stop services
up                 Create and start containers
migrate-to-labels  Recreate containers to add labels
```

In particular, the `logs` function is useful to prompt the logs from one or more components:

```
$ docker-compose logs web db
```

The `run` function can be used to run a command inside an isolated container that can be linked with the application ones, for example a PostgreSQL shell can be launched from a container linked to `db` for data manipulation:

```
docker-compose run --rm web psql
```

The psql command has to be available inside the web container and can be installed from Dockerfile adding:

```
RUN apt-get install -y postgresql-client
```

With the same method, a bash shell can be launched inside a container:

```
docker-compose run --rm web bash
```

The `--rm` option is equal to the `--rm` option of `docker run` command.

3.3 Gasista Felice architecture

Gasista Felice was initially created as a *Python/Django* application with a *jQuery* based interface. The evolution of technologies in web application clients and the diffusion of the mobile devices led to the necessity of a new mobile responsive web interface. Gasista Felice now provides a new interface based on *AngularJS* framework by Google, and the old jQuery interface referred as *legacy* user interface. *Nginx* web server is used as the application entry point. The role of Nginx consists in:

- routing the requests through the application components
- managing request buffering and queueing
- managing cryptography (https/tls)
- managing decompression of incoming requests and compression of responses
- caching the responses for reuse (disabled in development environments)

The routing of requests (Figure 3.3) consists in the following rules:

- requests related to the new user interface are forwarded to *HarpJS* server using the http protocol
- requests related to the REST API or the old user interface are forwarded to *uWGSI* using the uwsgi protocol

HarpJS is a static file server with built-in preprocessing and its role consists in serving HTML, CSS and Javascript files, that can be served directly or converted on request from higher abstraction level languages such as:

- *Markdown*, *Jade* and *EJS* for HTML
- *Sass*, *Less*, *Stylus* for CSS
- *Coffescript* for Javascript

uWSGI is an application server and its role consists in:

- starting and managing Python/Django processes
- forwarding the requests to the processes
- serving static files for the legacy interface

The containers for the Gasista Felice application (Figure 3.4) are:

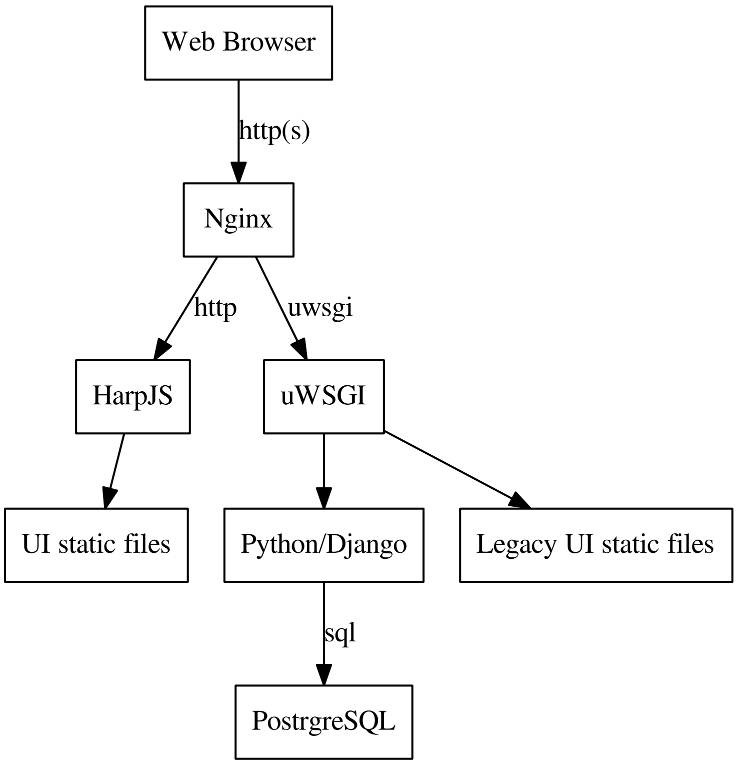


Figure 3.3: Requests routing for Gasista Felice application

- `proxy`: Nginx container
- `back`: uWSGI, Python/Django container
- `front`: HarpJs container
- `db`: PostrgreSQL container

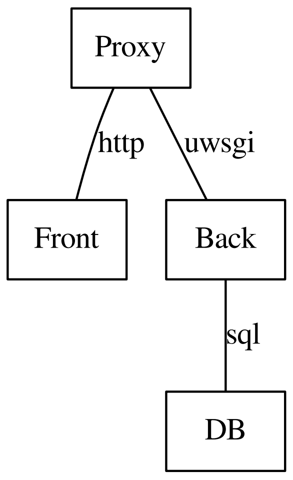


Figure 3.4: Gasista Felice containers and their interaction

The `docker-compose.yml` used for the development of Gasista Felice is:

```
proxy:
  image: befair/gasistafelice-proxy:latest
  volumes:
    - ./proxy/site.conf.dev:/etc/nginx/conf.d/site.conf:ro
  ports:
    - '127.0.0.1:8080:8080'
    - '127.0.0.1:8443:8443'
  links:
    - front
    - back

front:
  image: befair/gasistafelice-front:latest
  volumes:
    - ./ui:/code/ui:rw

back:
  image: befair/gasistafelice-back:latest
  volumes:
    - ./gasistafelice:/code/gasistafelice:ro
    - ./gasistafelice/fixtures:/code/gasistafelice/fixtures:rw
    - /tmp/gf_tracebacker:/tmp/tracebacker:rw
    - /tmp/gf_profiling:/tmp/profiling:rw
  ports:
    - '127.0.0.1:7000:7000'
  links:
    - db
  env_file: ./compose/settings.env

db:
  image: postgres:9.4
  env_file: ./compose/settings.env
```

For the `proxy` component, the configuration file `./proxy/site.conf.dev` is mounted to `/etc/nginx/conf.d` in read-only mode, the `8080` (http) and `8443` (https) ports are exposed, `front` and `back` containers are linked in order to allow Nginx to connect to the application frontend and backend.

For the `front` component, the directory containing the source code of the AngularJS interface is mounted inside `/code/ui` in read-write mode to be served by HarpJS. Write permissions on the file system are required in order to allow file processing by HarpJS.

For the `back` component, source code and fixtures are mounted inside the container, and the `7000` is exposed from uWSGI to enable direct http connection for debug purposes. The backend is linked to `db` container to access database features and the `settings.env` file is used for instancing environment variables for application configuration.

For the `db` component, the `settings.env` file is used for environment variables configuration. In order to start the Gasista Felice application:

```
$ git clone https://github.com/befair/gasistafelice
$ cd gasistafelice
$ git checkout master-dj17
$ make up
```

The following command can be used to load the test database:

```
$ make dbtest
```

The application is visitable at `http://localhost:8080`. The used images are hosted on Docker Hub. The Dockerfiles used for the images, as well as the configuration files `site.conf` used for Nginx configuration, `settings.env` and `Makefile` can be found in the Appendix.

Chapter 4

End-to-end testing

End-to-end testing is the practice of testing the software from the user's point of view. In a web application, an efficient approach to the end-to-end testing is obtained with the implementation of browser automation scripts, that involve the programmatic drive of one or different web browsers in order to access the application as client and test the features provided by its user web interface.

End-to-end tests are not sufficient to ensure the quality of the software: in order to offer a wider coverage of the software functionalities, unit tests and integration tests have to be implemented as well, but end-to-end tests offer the ability to discover bugs in user interface as well as bug in core components of the application: if an operation provides an unexpected result for the user, or triggers a server internal error, the causes of that behaviour have to be researched in the related core components of the software. End-to-end testing helps also to discover the bugs that are more visible to the final user, thus the most urgent.

In addition to this, end-to-end tests can be used in Test Driven Development to better define the specifications of the application. In fact, end-to-end testing objective is to ensure that the application specifications are satisfied, and can be used as a demonstration for the client that required features are implemented correctly.

This chapter will focus on the implementation of web automation based end-to-end tests with the Protractor framework for the Gasista Felice AngularJS web application. The key principles of AngularJS framework are explained in order to permit their exploitation trough Protractor. In fact, the reason that lead to choosing Protractor are its integration with the constructs of AngularJS framework: Protractor supports Angular-specific locator strategies, which allows to test Angular-specific elements without any setup effort [2]. At last, will be showed how to repeat the same test routines more times with different parameters, and how a mobile responsive interface can be tested with different window sizes.

While unit tests can be launched without additional configurations of the software, end-to-end tests require a more complex configuration, that will be discussed in detail in the next chapter.

4.1 Gasista Felice end-to-end testing

Web automation scripts that make use of Protractor have to be written in the Javascript programming language. In particular, the scripts used as specifications for the end-to-end tests to perform should contain the following constructs:

```
describe('Name of application or component', function() {
  it('should have feature X', function () {
    // test code for feature X
    ...
  });

  it('should perform operation Y', function() {
    // test code for operation Y
    ...
  });

  ...
});
```

The `describe()` function provided by Protractor framework can be used in order to describe the entire application or one of its components. The prototype of the `describe()` function is:

```
describe(string, function)
```

The name of the application or component is used as first parameter, and an anonymous function with the specification code is used as second parameter. The content of the anonymous function is a sequence of `it()` functions that are used to describe the test code for the features and operations of the application or component. The `it()` prototype is:

```
it(string, function)
```

A description of the feature or operation to test has to be provided as first parameter, and an anonymous function with the implementation of the test routine is provided as second parameter.

For Gasista Felice, the `test/e2e/spec.js` script is used as specification:

```
describe('GF ng-app', function() {
  ...
});
```

4.1.1 Connecting to the application

The first step for Gasista Felice testing consists in assuring that the application is up, connecting to the home page and inspecting the title displayed by the browser:

```
it('should have a title', function() {
  browser.get('http://proxy:8080/');
  expect(browser.getTitle())
    .toEqual('Gasista Felice');
});
```

In this snippet of code, the web browser performs a GET request to `http://proxy:8080` in order to access the home page of Gasista Felice, and checks that the title is the expected. The `get()` method of the `browser` object can be used to perform the request:

```
browser.get(url);
```

In order to check the title, the `expect()` method is used. The method take as input an object and returns another object with a `toEqual()` method that can be called to ensure that the object provided as its input is equal to the object provided as input for the `expect()` method:

```
expect(object1).toEqual(object2);
```

This construct is used by the Protractor framework as assertion: the test fails if the two object aren't equal.

4.1.2 Login

The next step consists in performing the login action and ensuring that the application redirects to the user's order page:

```
it('should connect to the user order page', function() {

  // fill login form
  element(by.model('app.username'))
    .sendKeys('01gas1');
  element(by.model('app.password'))
    .sendKeys('des');

  // click on 'GO' button!
  $$('#go').click();

  // check current url
  expect(browser.getLocationAbsUrl())
    .toBe('/order/');

  // check user displayed name
  expect(element(by.binding('person.display_name'))
    .getText())
      .toBe("Gasista_01 DelGas_01");
});
```

The username and password fields are retrieved from the page DOM by its model name. Model is a feature provided by the AngularJS framework that permits the two-way binding between a value of a DOM element and a object in the AngularJS application scope. Two-way binding means that if the value of the object is changed in the application scope the change reflects on the related DOM element and viceversa [16]. The definition of username and password elements for Gasista Felice is:

```
<input ng-model="app.username" id="username"
       type="text" class="validate">
<input ng-model="app.password" id="password"
       type="password" class="validate">
```

The `ng-model` directive permits the binding of a DOM element to a property of the AngularJS scope [17]. This is an example of how Protractor is designed to work with AngularJS constructs. Css selectors can be used as well but assuming that the model of the username and password fields will never be changed, the model name provides a more reliable reference to these elements. However, sometimes the use of a css selector is the only way to select a DOM element, such as for the `GO` button of Gasista Felice login page. The `$$(css_selector)` function is an alias for:

```
element(by.css(css_selector))
```

The `$$()` function is used to select an element of the DOM through a css selector. In the code above the `GO` button is accessed using `#go` selector and clicked through its `click()` method. The button definition is:

```
<a ng-click="app.login()" href="#" id="go"
   class="waves-effect waves-blue-grey
          waves-grey-blue btn modal-action modal-close">
   GO
</a>
```

The `#go` css selector refers to the id of the button, defined using `id="go"`. One of the button classes could be used as well, but the id is preferable (assuming that it's unique in the page context).

After the login, the `getLocationAbsUrl()` method of the `browser` object is used to check that the current url matches the one related to the orders page.

The `person.display_name` binding is used as reference to the DOM element with the function of displaying the person name and its `getText()` method is used to retrieve the text to verify with the `expect()` function. Binding is another feature of the AngularJS framework that permits the one-way binding between the application scope and the DOM element. One-way binding means that if the value of the object is changed in the application scope the change reflects on the DOM element, but a change in the DOM element doesn't reflect on the application scope. In order to perform a one-way binding the `ng-bind` [18] or double curly brackets can be used:

```
<p ng-bind="expression"></p>
<p>{{expression}}</p>
```

The related DOM element definition is:

```
<a ng-link="profile()" href=""
   title="Il mio profilo">
   {{ person.display_name }}
</a>
```

4.1.3 Purchasing management

Once the user is logged and the browser is located in the order page, the purchasing management is tested. The first action to perform is trying to increment and decrement the quantity for a product and verify that the displayed quantity and related total price is correct:

```
it('should increment/decrement the price/qty when "+/-" are clicked',
   function () {

  // get the second item in the table
  var item = element.all(
               by.repeater('product in order.pm.products')
             ).get(1);

  // click 20 time on '+'
  for (var i=0; i < 20; i++)
    item.$$('.glyphicon-plus')
      .click();

  // click 10 times on '-'
  for (var i=0; i < 10; i++)
```

```
    item.$$('.glyphicon-minus')
      .click();

  // qty should be 10 units
  expect(item.element(by.model('product.quantity'))
    .getAttribute('value'))
      .toBe('10');

  // price should be 250 euros
  expect(item.element(by.binding('product.quantity'))
    .getText())
      .toBe('€ 250,00');
});
```

To retrieve the product from the list, the following construct is used:

```
element.all(by.repeater(repeater)).get(index);
```

Repeater is another of the features of the AngularJS framework. In particular, the `ng-repeat` directive permits to generate DOM elements using an html template and an array of objects [19]. In Gasista Felice the list of products is generated using the `ng-repeat` directive with `product in order.pm.products` as repeater, so the same repeater can be used to retrieve the products in Protractor. In the code above the second product is retrieved from the list and the increment and decrement buttons are retrieved using its css class. Even if there are more buttons (a pair for every product) that share the same class, the right pair is retrieved because the `$$()` is called as method of the `item` object. The buttons are clicked multiple times using `for` loops.

Product quantity and price are retrieved using model selector and binding selector, respectively, and again, the methods are called from the `item` object to exclude other products. A particular attention have to be paid at this instruction:

```
  expect(item.element(by.binding('product.quantity'))
    .getText())
      .toBe('€ 250,00');
```

The `product.quantity` binding is used in order to retrieve the total price for the product. The relation between the used binding and the total price of the product has to be researched in its DOM element definition:

```
<td data-title="total" class="tdprod" >
  {{product.quantity*product.price | currency:"€"}}
</td>
```

AngularJS framework permits to perform operations in the template, in this case the total price is calculated as product between unitary price and quantity. In a *MVC* (Model View Controller) software architecture, the business logic should be separated from the view, and performing operations in the template violates this principle. This is an example of how writing tests can lead to the discovery of bad code design choices.

Another check to perform is that the quantity for a product should never go under zero:

```
it('should never decrement the price/qty under 0',
   function () {

  var item = element.all(
               by.repeater('product in order.pm.products')
             ).get(1);

  for (var i=0; i < 20; i++)
    item.$$('.glyphicon-minus')
      .click();

  expect(item.element(by.model('product.quantity'))
    .getAttribute('value'))
      .toBe('0');

  expect(item.element(by.binding('product.quantity'))
    .getText())
      .toBe('€ 0,00');
});
```

The second product is retrieved from the list, the decrement button pressed 20 times and the quantity value checked.

4.1.4 Insertion and check of products in the basket

In order to check the insertion of the products in the basket, the following test is used:

```
it('should add a product to the basket',
   function() {

  var item = element.all(
                by.repeater('product in order.pm.products')
             ).get(2);

  // set the quantity to 3
```

```
item.element(by.model('product.quantity'))
  .clear();
item.element(by.model('product.quantity'))
  .sendKeys('3');

// add to the basket
element(by.buttonText('Aggiungi al paniere'))
  .click();

// handle the alert popup
handleAlertPopup();

// go to the basket
browser.setLocation('basket');

// get the first order
item = element.all(
         by.repeater('item in basket.open_ordered_products')
       ).get(0);

// get all the column from the first order
var columns = item.$$('td');

// expects to have 8 columns (counting the hidden ones)
expect(columns.count())
  .toBe(8);

// check the fields
expect(columns.get(0)
  .getText())
    .toBe('Ord. 59');
expect(columns.get(1)
  .getText())
    .toBe('Fornitore 01');
expect(columns.get(2)
  .getText())
    .toBe('Scarpe Uomo (paio)');
expect(columns.get(3)
  .getText())
    .toBe('€ 20,00');
expect(item.element(by.model('item.quantity'))
  .getAttribute('value'))
    .toBe('3');
expect(columns.get(6)
    .getText())
      .toBe('€ 60,00');
});
```

The third product is retrieved from list, then its quantity field is cleared and set to 3, the `Aggiungi al paniere` button is retrieved from its text and clicked to add the product to the basket, and the resulting alert is managed with the `handleAlertPopup()` function:

```
var handleAlertPopup = function() {
  var EC = protractor.ExpectedConditions;
  browser.wait(EC.alertIsPresent(), 5000);
  browser.switchTo().alert().accept();
};
```

Then the location is set to the basket page trough the `setLocation()` method of the `browser` object, the list of the ordered products is retrieved and for the first item in the list the fields are counted and checked. As can be seen from the code snippet above, a selection method such as `$$()` can return more than one element, and in that case, the elements can be counted with the `count()` method and individually accessed providing an index to the `get()` method. As `getText()` can be used to get the text of a DOM element, the `getAttribute()` method can be used to get the value of the DOM element attribute provided as parameter.

4.1.5 Testing parametrization and mobile responsive applications

Gasista Felice is a mobile responsive application, so the implemented tests have to be applied two times with different window dimensions.

In order to repeat two times the tests the `map()` method of `Array` objects can be exploited. *The map() method creates a new array with the results of calling a provided function on every element in this array* [27]. With the `map()` method an array of indexes can be created and these indexes can be used as parameter for the execution of test routines:

```
[0,1].map(function(index) {
  it('should have a title', function() {
    if (!index)
      browser.driver.manage().window().maximize();
    else
      browser.driver.manage().window().setSize(768, 1024);

    browser.get('http://proxy:8080/');
    expect(browser.getTitle())
```

```
      .toEqual('Gasista Felice');
  });

  ...
});
```

In this way the routines are performed two times, the first time with a maximized window (the size of the maximized window depends on the desktop resolution used in the operating system that runs the browser), the second time with a fixed size that triggers the mobile interface. The `index` variable can always be checked in order to determine the current interface. More indexes can be used in order to repeat the tests with different window sizes.

The main difference between the Gasista Felice desktop and mobile interface is that the second make use of a navigation bar that display user related informations and logout and settings buttons, while in the desktop interface these elements are always showed. This navigation bar can be showed and hided with the click of its toggle button.

Button toggle can be implemented with simple if-else instructions:

```
...

it('should connect to the user order page', function() {
  ...

  // check user displayed name
  if (index) {
    $$('.navbar-toggle').click();
    expect(element(by.binding('person.display_name'))
      .getText())
        .toBe("Gasista_01 DelGas_01");
    $$('.navbar-toggle').click();
  }
  else
    expect(element(by.binding('person.display_name'))
      .getText())
        .toBe("Gasista_03 DelGas_01");
});

...
```

The button for showing and hiding the navigation bar is retrieved from its css class and clicked twice (first time to show, second time to hide again).

The `map()` method can also be used from an array of objects:

```
var getElementByIndex = function(index) {
    // implentation
    ...
    return item;
}

[{index:1, foo: 1, bar:2}, {index: 2, foo:2, bar:1}]
  .map(function(obj) {
    it('...', function() {
      item = getElementByIndex(obj.index);
      expect(item.foo)
        .toBe(obj.foo);
      expect(item.bar)
        .toBe(obj.bar);
    });
    ...
  });
```

This technique permits to run test routines more times with different sets of values.

Logout

Because with the use of `map()` the login routine is performed twice, a logout routine has to be appended to the bottom of test specifications:

```
it('should logout', function() {
  if(index)
    $$('.navbar-toggle').click();
  $$('#btn-logout').click();
});
```

Chapter 5

Setup of the test environment

While in the previous chapter the Protractor framework has been used in order to implement browser automation based end-to-end tests for the AngularJS web interface of Gasista Felice application, in this chapter the software used to run Protractor based end-to-end tests is discussed and in particular its container based virtualization and interaction with the web application containers.

Given that Docker and Docker Compose are already used to run the containerized application, the container structure is extended in order to include the testing containers and to permit the running of the end-to-end tests without the burden of manual installation and configuration of the testing framework resting on developers.

5.1 Protractor

Protractor is not available yet as official Docker image. With the purpose of running Protractor end-to-end tests for Gasista Felice and other AngularJS web applications, a Protractor image has been builded and pushed to the Docker Hub. The image is called `michelesr/protractor`, and this is the Dockerfile used for the build:

```
FROM iojs:3

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

RUN npm install -g protractor

RUN mkdir /code

WORKDIR /code

CMD ["protractor", "conf.js"]
```

Protractor is a *Node.js* application. Node.js is a cross-platform runtime environment for server-side and networking Javascript applications. In order to run Protractor, an *Io.js* image has been used as base. Io.js is a fork of the Node.js open-source software, created for the primary purpose of moving the Node.js project to a structure where the contributors and community can step in and effectively solve the problems facing Node (including the lack of active and new contributors and the lack of releases) [14]. From the `iojs:3` base image, Protractor framework is installed using `nmp` (Node Package Manager), the `/code/` directory is created and used as working directory, and `protractor conf.js` is used as default command for running the framework.

The `conf.js` configuration file is the entry point of Protractor:

```
// conf.js
exports.config = {
  seleniumAddress: 'http://hub:4444/wd/hub',
  specs: ['spec.js'],
  multiCapabilities: [
    { browserName: "firefox" },
    { browserName: "chrome" }
  ]
}
```

The `exports.config` object is used to define the configuration of the Protractor framework, that includes:

- a list of the test routine specification files
- a list of the browsers used as clients for the web application
- the network address of the *Selenium* server

5.2 Selenium

Selenium is an application framework for the execution of browser automation scripts, and is written in the Java programming language.

The history of Selenium begin with *Selenium Remote Control*. Selenium RC is a server that accept requests for browser automation through an HTTP API. The HTTP API can be accessed with drivers available for different programming language, such as Java, Python, PHP, Ruby, .NET, Perl and Javascript. With the diffusion of Selenium and browser automation scripts, browsers start to provide their native support to automation, leading to the creation of *Selenium Web Driver*.

Selenium Web Driver is the successor of Selenium RC, and the main difference consists in its interaction with web browsers, implemented with browser specific drivers. While in Selenium RC a server for running tests is required, Selenium Web Driver directly starts and controls the browsers. Client API for Selenium Web Driver are available for different languages, so that the automation scripts can be written in the developer preferred language. In order to run Web Driver on remote machines a Selenium Grid server is required. Selenium Grid is divided in different components:

- a Selenium server, called *hub*, that serves as controller for the browsers
- one or more web driver nodes for the required browsers (Firefox and Chrome)

The Web Driver nodes attach to the Selenium hub in order to be controlled, then the Protractor testing framework sends request to the hub in order to run end-to-end tests in the desired browsers. The hub search the required browsers from the available nodes and controls them in order to test the web application.

The nodes can be located in different machines and operating systems. In particular, in this implementation the hub and the nodes runs inside Docker containers, and they are linked through the Docker VPN (Figure 5.1). Like Selenium RC, the Selenium Grid hub is accessible through the HTTP API.

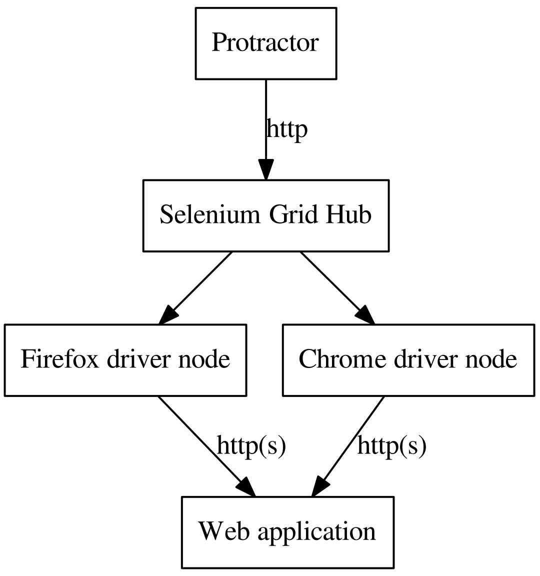


Figure 5.1: Interactions between Protractor and Selenium ecosystem

5.3 Containers configuration

The containers used for end-to-end testing purpose are:

- `hub`: Selenium Grid hub
- `firefox`: Selenium Grid node for Mozilla Firefox browser
- `chrome`: Selenium Grid node for Google Chrome browser
- `e2e`: Protractor framework

In order to define the testing containers and their interaction with the application (Figure 5.2), a new Docker Compose configuration file is provided:

```
hub:
  image: selenium/hub:latest

firefox:
  image: selenium/node-firefox-debug:latest
  links:
    - hub
    - proxy
  ports:
    - '127.0.0.1:5900:5900'
  env_file:
    - ./test/e2e/settings.env

chrome:
  image: selenium/node-chrome-debug:latest
  links:
    - hub
    - proxy
  ports:
    - '127.0.0.1:5901:5900'
  env_file:
    - ./test/e2e/settings.env

e2e:
  image: michelesr/protractor:latest
  volumes:
    - ./test/e2e:/code:ro
  links:
    - hub
```

In particular, the `firefox` and `chrome` containers are linked to `hub` for registering and to `proxy` that acts as entry point of the web application. The `e2e` tests is linked with the `hub` in order to allow the forwarding of test requests. This new configuration file is named `compose/test.yml` and is used to run the tests trough the Makefile:

```
...

test-cat.yml: docker-compose.yml compose/test.yml
	@cat docker-compose.yml compose/test.yml > test-cat.yml

...

test-e2e: test-cat.yml
	@echo 'End-to-end test: running protractor'
	@docker-compose -f test-cat.yml up -d
	@sleep 5
	@docker-compose -f test-cat.yml run --rm e2e

...
```

The `test-cat.yml` file is generated as concatenation of `docker-compose.yml` and `compose/test.yml` and is used by the `test-e2e` role of the Makefile as configuration file of Docker Compose. The `@` is used as command prefix to avoid their printing on the console. The `sleep 5` is used to wait 5 seconds after containers start in order to made their processes initiate correctly before sending requests to them.

5.3.1 Test Running

The end-to-end tests can be launched from Gasista Felice repository root with:

```
$ make up
Starting gasistafelice_front_1...
Starting gasistafelice_db_1...
Starting gasistafelice_back_1...
Starting gasistafelice_proxy_1...

$ make dbtest
...
...

$ make test-e2e
End-to-end test: running protractor
Creating gasistafelice_hub_1...
```

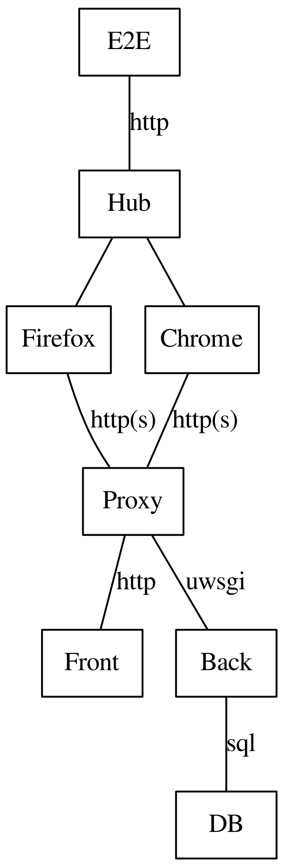


Figure 5.2: Linking of testing containers with application containers

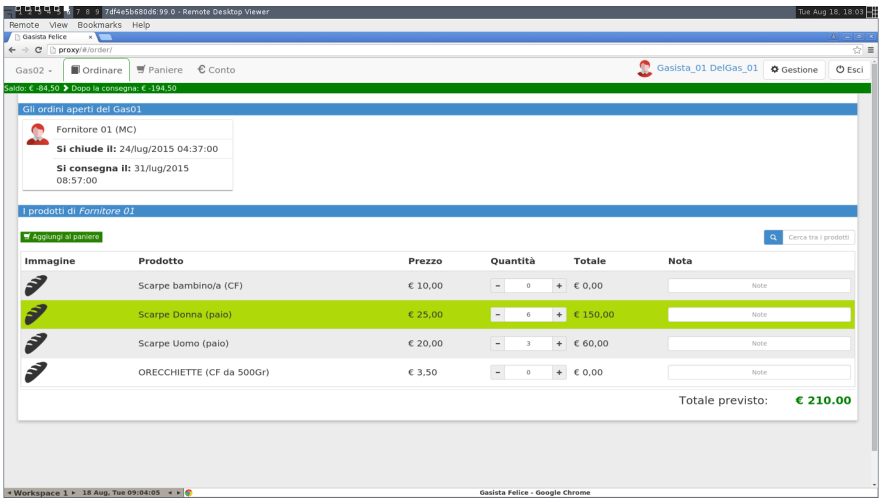
```
gasistafelice_db_1 is up-to-date
gasistafelice_back_1 is up-to-date
Creating gasistafelice_e2e_1...
gasistafelice_front_1 is up-to-date
gasistafelice_proxy_1 is up-to-date
Creating gasistafelice_chrome_1...
Creating gasistafelice_firefox_1...
[launcher] Running 2 instances of WebDriver
............
------------------------------------
[chrome #2] PID: 15
[chrome #2] Specs: /code/spec.js
[chrome #2]
[chrome #2] Using the selenium server at http://hub:4444/wd/hub
[chrome #2] ............
[chrome #2]
[chrome #2] Finished in 17.822 seconds
[chrome #2] 12 tests, 28 assertions, 0 failures
[chrome #2]

[launcher] 1 instance(s) of WebDriver still running
........
------------------------------------
[firefox #1] PID: 10
[firefox #1] Specs: /code/spec.js
[firefox #1]
[firefox #1] Using the selenium server at http://hub:4444/wd/hub
[firefox #1] ............
[firefox #1]
[firefox #1] Finished in 21.79 seconds
[firefox #1] 12 tests, 28 assertions, 0 failures
[firefox #1]

[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #2 passed
[launcher] firefox #1 passed
```

The images used for `hub`, `firefox` and `chrome` containers, provided by Selenium developers, are retrieved from Docker Hub. The `michelesr/protractor` image, retrieved from Docker Hub, has been built using the Dockerfile exposed previously.

5.3.2 Inspect browsers behaviour through VNC

The `selenium/node-firefox-debug` and `selenium/node-chrome-debug` are distributed with a built-in VNC server that can be accessed in order to visually inspect the browser behaviour during the running of end-to-end tests. For this purpose the `compose/test.yml` exposes ports `5900` of `firefox` and `chrome` containers as `5900` and `5901`, so they can be accessed with a VNC client (Figure 5.3). The environment configuration file `test/e2e/settings.env` can be used to set the screen resolution used by the VNC servers:

```
SCREEN_WIDTH=1920
SCREEN_HEIGHT=1080
```



Saldo: € -84,50 › Dopo la consegna: € -194,50

Gli ordini aperti del Gas01

Fornitore 01 (MC)
Si chiude il: 24/lug/2015 04:37:00
Si consegna il: 31/lug/2015 08:57:00

I prodotti di *Fornitore 01*

Immagine	Prodotto	Prezzo	Quantità	Totale	Nota
	Scarpe bambino/a (CF)	€ 10,00	0	€ 0,00	Note
	Scarpe Donna (paio)	€ 25,00	6	€ 150,00	Note
	Scarpe Uomo (paio)	€ 20,00	3	€ 60,00	Note
	ORECCHIETTE (CF da 500Gr)	€ 3,50	0	€ 0,00	Note

Totale previsto: **€ 210.00**

Figure 5.3: Google Chrome browser running inside the chrome container

Chapter 6

Continuous Integration

6.1 Overview

Continuous Integration is a software development practice where members of a team integrate their work frequently: each person integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible [24]. This approach is the opposite of Deferred Integration, where the work of the developers is integrated less frequently, usually leading to integration problem. In Deferred Integration the integration is treated as an event, while in Continuous Integration is a daily process. Continuous integration is one of the twelve practices of *XP* (*Extreme Programming*) [11], and is essential in order to adopt an Agile software development process inside an organization.

In Continuous Integration, a developer is always aligned with the changes introduced by other developers, and every change is tested by a Continuous Integration system as soon as committed and pushed in the Source Code Management system (SCM). Continuous integration helps to spot conflicts between the work of two or more developers, allowing fast resolution, and therefore avoiding the waste of a huge amount of time. Spotting the bugs as soon as possible helps their localization and fix, while in Deferred Integration bugs are cumulative and hard to manage.

The main benefits of Continuous Integration are the reduction of the integration times and efforts (that involve economical costs), and the ability to always release a new working version of the product (frequent small releases is also an *XP* practice). Another benefit of Continuous Integration is the communication: implementing an automated building and testing system will give visibility of the current status of a project to all the person involved in his realization, thus avoiding misunderstandings between those persons. Build results can also be published and act as a quality indicator of the software development process and released products. For Open Source projects, a Continuous Integration system provides an incentive for contribution.

In order to provide an efficient Continuous Integration service, the CI system has to satisfy the following requirements:

- the build process that follows a push from the developer has to be fast to provide the feedback as soon as possible
- a daily or weekly automated complete build of the projects has to be performed in order to assure the correctness of the software and the compatibility with its external dependencies
- the environment used for the build has to be similar to the environment used in production

The implementation will cover those aspect. The *build* term is used to indicate the action of retrieving the application from SCM, building it and running the automated tests.

6.1.1 Forking Workflow for the Development

A common approach to team development inside an organization consists in setting a main repository for the project, also referred as *upstream*: all the developers that contribute to the project fork the main repository in their account on the SCM server, clone the repository in their local machines, push the changes on their personal fork on the server and then make a pull request on the upstream for changes review and merge. If a forking workflow is adopted, is necessary that all the forks that are pushed in SCM are also tracked by the Continuous Integration system.

The Continuous Integration good practices involve the pulling of last changes, their integration and the running of automated tests before every developer commit, in order to discover the conflicts between the upstream and the local work of the developer.

For `gasistafelice`, the main repository and developers forks are located on Github server. The fork of the repository that acts as a use case in this chapter is available at `https://github.com/michelesr/gasistafelice`. The `dev` branch contains the latest change introduced by the developer, while the `master` branch is aligned with the upstream.

6.1.2 Organization

This chapter presents the organization and implementation of a Continuous Integration system. This implementation is based on the cooperation between different components:

- a Git based Source Code Management
- a Continuous Integration tool

- Docker and Docker Compose

For every component used in the implementation, the installation and configuration procedure is explained. The procedure can be reproduced in desktop environments as well as in servers. Before the description of the procedure, the configuration of the Gasista Felice containers is presented.

6.2 Gasista Felice containers configuration

Gasista Felice is configured, through `docker-compose.yml`, to pull the images for the application components from Docker Hub, and to mount the source code of the application from the local repository to allow changes to be reflected inside the container. The component Dockerfiles are designed to always perform a COPY instruction to copy the source code from the repository during the build, that is performed by Docker Hub, but given that `docker-compose.yml` contains mount instructions, the content copied at build time is replaced with the content of the repository that resides in the developer host (this is the normal behaviour in a Unix-like system where during the mount the content of the directory specified as mount point is replaced with the content of the mounted file system).

The mounting approach is perfect for development, where every change to the code has to be applied immediately without the rebuild of the images, but in production the code is copied when the images are built and changes imply rebuilding. To provide a Continuous Integration testing environment closer to the one used in production, a new `docker-compose.yml` is required:

```
proxy:
  build: ./proxy
  ports:
    - '127.0.0.1:8080:8080'
    - '127.0.0.1:8443:8443'
  links:
    - front
    - back

front:
  build: ./ui

back:
  build: ./gasistafelice
  links:
    - db
  env_file: ./compose/settings_ci.env
```

```
db:
  image: postgres:9.4
  env_file: ./compose/settings_ci.env

hub:
  image: selenium/hub:latest

e2e:
  build: ./test/e2e
  links:
    - hub

firefox:
  image: selenium/node-firefox-debug:latest
  links:
    - hub
    - proxy
  env_file:
    - ./test/e2e/settings.env

chrome:
  image: selenium/node-chrome-debug:latest
  links:
    - hub
    - proxy
  env_file:
    - ./test/e2e/settings.env
```

In this `docker-compose.yml` the `image` declarations are replaced with `build`, and path for the Dockerfile with the build instructions are provided for every application component image. To avoid conflicts with the original Docker Compose file, this new configuration file is called `compose/ci.yml`. The ports used for debugging purposes are removed.

The `settings.env` has been replaced with `settings_ci.env` in some components to override the environment variables used in development with the production variables. The discussed changes has been applied to the Gasista Felice repository.

6.2.1 Docker caching system

In order to reduce build times, providing a faster feedback to the developer, the Docker caching system is exploited. When Docker builds an image from a Dockerfile, if it founds an image layer already produced for that instruction, it avoids the recreation of that layer. The cache is invalidated when the Dockerfile changes, or for the COPY instructions, when the content inside the directory to copy changes.

If the content of the directory to copy inside the container change, then the cache for the COPY instruction will be invalidated, and the instruction will be executed again, leading to a different output layer. If the layer produced by the COPY instruction is different, than the cache is invalidated for all the following instruction in the Dockerfile.

To avoid the rebuilding of the entire image, including software dependencies that are downloaded through package managers, the COPY instructions for copying the source code of the component inside the container have to be placed at the bottom of the Dockerfile, encouraging Docker to not rebuild all the layer related to the dependencies.

6.3 Gogs - Go Git Service

Gogs (Go Git Service) is an open source lightweight Git Service that plays the role of *SCM* (Source Code Management) in the Continuous Integration implementation. Like Docker, Gogs is written in the Go programming language, that is a compiled language. Go programs can be compiled in different operating systems (including Linux, Mac OS and Windows) and the result of the compilation is always a single binary, due to its static linking policy for libraries. Thus, Gogs can be run as a standalone binary or inside a Docker container, that is the choice made for this implementation.

6.3.1 Installation

In order to install Gogs, a prebuilt image from Docker Hub is used:

```
$ docker pull gogs/gogs
$ docker run --name gogs -d \
      -p 3000:3000 \
      -p 5000:8080 \
      -p 32:22 \
      -v $HOME/gogs_data:/data \
      gogs/gogs
```

The ports 3000, 8080 and 22 are exposed respectively to 3000 and 5000 and 32of the host. The port 3000 (http) is reserved for the Gogs service, while the 8080 (http) is used for the Continuous Integration system, that will be linked with Gogs later, and 22 is for ssh, that is used to establish secure connections to git repositories. Gogs require a directory for configuration files and data storage, so `$HOME/gogs_data` has been mounted inside the container as `/data`.

6.3.2 Configuration

The opening of the browser at `http://localhost:3000/` redirects to the first configuration page. The configurations to adopt are showed in Table 6.1 and Table 6.2 .

Table 6.1: Gogs initial configuration

Option	Value
Database type	`SQLite3`
Database path	`data/gogs.db`
Application Name	`Gogs: Go Git Service`
Repository Root Path	`/home/git/repositories`
Run User	`git`
Domain	`localhost`
SSH Port	`22`
HTTP Port	`3000`
Application URL	`http://localhost:3000/`

Table 6.2: Admin settings for Gogs

Option	Value
Username	`SuperUser`
Password	`*********`
Email	`superuser@example.org`

Settings are confirmed with `Install Gogs` button. Then a normal user account has to be registered (Table 6.3) through the `Register` button.

Table 6.3: User registration in Gogs

Option	Value
Username	`mike`
Email	`mikefender@cryptolab.net`
Password	`*********`

6.3.3 SSH Configuration

When the user connects to the repositories using http or https protocol, the authentication is password based. Password can be lost or stolen, and an asymmetric key based authentication is more secure. Also if http is used, the connection isn't ciphered and sensible data can be sniffed from the network.

When the user register a public key for his account, the key can be used to authenticate him on repository accesses through ssh protocol. In order to generate a key pair, the user can run this command in his machine:

```
$ ssh-keygen -C "ECDSA key for Gogs service" \
            -f id_ecdsa_gogs -t ecdsa -b 256

Generating public/private ecdsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_ecdsa_gogs.
Your public key has been saved in id_ecdsa_gogs.pub.
The key fingerprint is:
SHA256:/j+EDsnTpW2F+JXXeF5IwqYIa7pG/5j+bZFgSMBgUAg
ECDSA key for Gogs service
The key's randomart image is:
+---[ECDSA 256]---+
|E++o..     .     |
|..  . o     + .  |
|     . + . + + +.|
|      + + o o * =|
|     o oS+ B o +.|
|    o  .= * =   .|
|   . o  .+ +     |
|    o .o oo .    |
|   . .+oo.o...   |
+----[SHA256]-----+
```

The command generates a 256 bit *ECDSA* (Elliptic Curve Digital Signature Algorithm) key pair. The ECDSA provides smaller key size than RSA algorithm for the equivalent level of security [22]. An optional passphrase can be added to lock and unlock the private key. The ssh client has to be configured to use the right key and port to access Gogs, through the `~/.ssh/config` file:

```
Host localhost
  User mike
  Port 32
  IdentityFile ~/.ssh/id_ecdsa_gogs
  IdentitiesOnly yes
```

The public key `id_ecdsa_gogs.pub` can be added to the Gogs server accessing to the user account settings and selecting `SSH Keys -> Add Key`. The key content has to be provided in the `Content` field, and an arbitrary name can be chosen, then the operation is confirmed with `Add Key` button.

6.3.4 Adding the Gasista Felice repository

After the registration and sign in, a repository for Gasista Felice named `gasistafelice` has to be created. After the creation, the local repository can be pushed on the Gogs server using git:

```
$ cd path/to/gasistafelice/
$ git remote add gogs git@localhost:mike/gasistafelice.git

$ git push gogs master
The authenticity of host '[localhost]:32 ([127.0.0.1]:32)'
can't be established.
ECDSA key fingerprint is
SHA256:agpFsLWSqB4UJG/W0VQDu5pRVIT2pmp7h+94IMYReec.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:32' (ECDSA)
to the list of known hosts.
Counting objects: 22310, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8108/8108), done.
Writing objects: 100% (22310/22310), 19.24 MiB | 18.89 MiB/s, done.
Total 22310 (delta 14168), reused 21654 (delta 13629)
To git@localhost:mike/gasistafelice.git
 * [new branch]      master -> master
```

The warning appears at the first connection to the server, then the public key of the server is added to `~/.ssh/known_hosts` and the message doesn't present again.

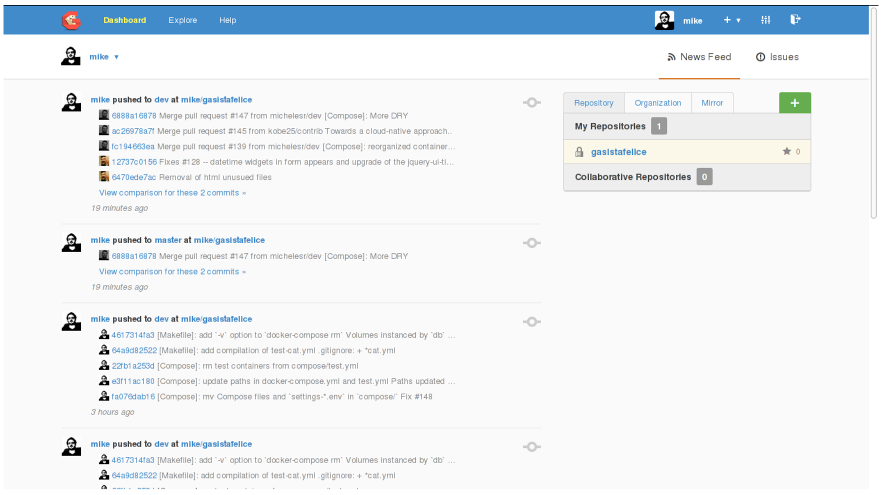
After the push, the repository can be managed trough git client and Gogs web interface (Figure 6.1).

6.4 Jenkins

Jenkins is a open source software for Continuous Integration written in Java. It permits the scheduling of periodical or triggered builds and SCM pools for Continuous Integration.

In order to build container based applications, Jenkins container require to access a Docker daemon, and three different approaches are available for the purpose:

- grant the access to the host Docker daemon inside the Jenkins container
- use https to communicate to the Docker daemon
- install Docker inside the Jenkins container



mike pushed to dev at mike/gasistafelice
6888a16878 Merge pull request #147 from michelesr/dev [Compose]: More DRY
ac26978a7f Merge pull request #145 from kobe25/contrib Towards a cloud-native approach..
fc194663ea Merge pull request #139 from michelesr/dev [Compose]: reorganized container...
12737c0156 Fixes #128 -- datetime widgets in form appears and upgrade of the jquery-ui-ti...
6470ede7ac Removal of html unusued files
View comparison for these 2 commits »
19 minutes ago

mike pushed to master at mike/gasistafelice
6888a16878 Merge pull request #147 from michelesr/dev [Compose]: More DRY
View comparison for these 2 commits »
19 minutes ago

mike pushed to dev at mike/gasistafelice
4617314fa3 [Makefile]: add `-v` option to `docker-compose rm` Volumes instanced by `db` ...
64a9d82522 [Makefile]: add compilation of test-cat.yml .gitignore: + *cat.yml
22fb1a253d [Compose]: rm test containers from compose/test.yml
e3f11ac180 [Compose]: update paths in docker-compose.yml and test.yml Paths updated ...
fa076dab16 [Compose]: mv Compose files and `settings-*.env` in `compose/` Fix #148
3 hours ago

mike pushed to dev at mike/gasistafelice
4617314fa3 [Makefile]: add `-v` option to `docker-compose rm` Volumes instanced by `db` ...
64a9d82522 [Makefile]: add compilation of test-cat.yml .gitignore: + *cat.yml

Figure 6.1: Gasista Felice repository page at Gogs

In term of security, the https based solution is the worst because exposes Docker daemon on the network, while the other two solution are on the same security level: sharing the Docker daemon on the host allows trusted Jenkins users to exploit it in order to mount host volumes with write permission, while installing and running Docker inside a container requires the creation of privileged containers that can access all the host features including devices and kernel specific functions. In every case, using a container for Jenkins is always better than installing Jenkins in the host system because Docker provides a layer of isolation from the system resources, and given that Jenkins is the only process running inside the container, the only way to launch a malicious script is from a Jenkins job. The creation of Jenkins jobs can be restricted to trusted users from the security configuration.

The first solution (Figure 6.2), consisting in the sharing of Docker daemon inside the Jenkins container has been adopted for the following reasons:

- it doesn't require the creation of privileged containers
- the same Docker daemon used to run Gogs and Jenkins is reused from Jenkins, avoiding overhead of running another Docker daemon
- using the same Docker daemon increases the chances to exploit cache
- the implementation is simple

6.4.1 Installation

In order to install Jenkins and the support software, a custom Dockerfile is required:

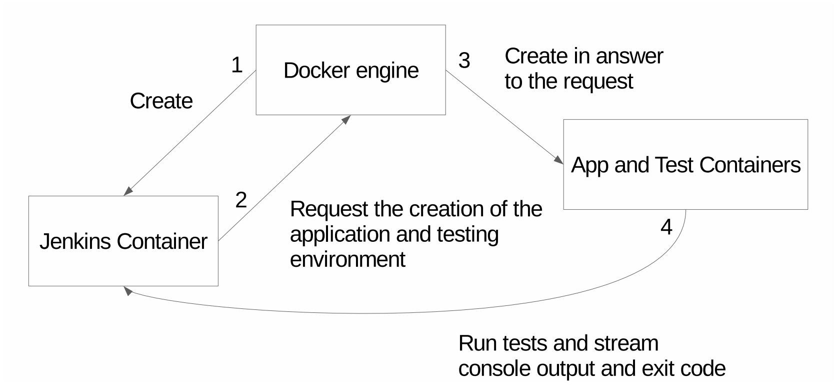

Docker engine
1
Create
3
Create in answer to the request
App and Test Containers
2
Jenkins Container
Request the creation of the application and testing environment
4
Run tests and stream console output and exit code

Figure 6.2: Jenkins interaction with Docker and testing environment

```
FROM jenkins:latest

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

USER root
RUN apt-get update \
      && apt-get install -y python python-pip sudo \
      && rm -rf /var/lib/apt/lists/*
RUN pip install 'docker-compose==1.4'
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers

USER jenkins
COPY plugins.txt /usr/share/jenkins/plugins.txt
RUN /usr/local/bin/plugins.sh /usr/share/jenkins/plugins.txt
```

In this Dockerfile, starting from a Jenkins base image, `python2`, `python-pip` (Python Package Manager), `docker-compose` and `sudo` packages are installed, then superuser privileges are granted to the Jenkins user in order to permit the use of Docker. Generally the Docker daemon can be accessed without superuser privileges if the user is added to the `docker` group, but given that Jenkins is running inside a container, `sudo` is necessary. The `plugins.txt` file containing the list of plugin to install copied inside the container is:

```
scm-api:latest
git-client:latest
git:latest
greenballs:latest
```

The command to properly run the Jenkins container, providing access to Docker daemon and connection to the Gogs network is:

```
$ docker pull michelesr/jenkins
$ mkdir $HOME/jenkins_data
$ docker run --name jenkins -d \
     -v /var/run/docker.sock:/var/run/docker.sock \
     -v /usr/lib/libdevmapper.so.1.02:/usr/lib/libdevmapper.so.1.02 \
     -v $(which docker):/usr/bin/docker \
     -v $HOME/jenkins_data:/var/jenkins_home \
     --net container:gogs \
     michelesr/jenkins
```

In order to grant access to the Docker daemon inside the container, the Unix socket `/var/run/docker.sock`, the Docker client (retrieved with the bashism `$(which docker)`), the `libdevmapper` library used for the creation of virtual volumes, and a directory for Jenkins data have to be mounted. With the `--net container:gogs` parameter, the Jenkins container will share the same network stack of the Gogs container, and they will be able to communicate through the loopback device `localhost`.

The command has been tested on an Arch Linux distribution and some parameters (such as the library path) can be different in another distribution. Also the `jenkins_data` directory has to belong to the same UID of the Jenkins user (`1000`). The UID of the current user can be retrieved with the command:

```
$ cat /etc/passwd | grep $(whoami)
michele:x:1000:1000::/home/michele:/usr/bin/zsh
```

If the UID doesn't match, it can always be forced with:

```
# chown 1000:1000 $HOME/jenkins_data -R
```

Assuming that Gogs is running and has been launched with the command provided previously, Jenkins can be accessed at the url `http://localhost:5000`, because the port for the Jenkins service has been exposed at the launch of Gogs container.

6.4.2 Security Configuration

In order to avoid the execution of malicious scripts in Jenkins, the creation and configuration of jobs has to be restricted to a small set of trusted users. In the page `http://localhost:5000/configureSecurity/` the `Enable security` checkbox has to be checked, and the options from Table 6.4 have to be setted.

Table 6.4: Security configuration for Jenkins

Option	Value
TCP Port for JNLP slave agents	`Disable`

Option	Value
Security realm	`Jenkins own user database`
Allow user to sign-up	`unchecked`

After confirming, an user has to be added accessing the page at the url: `http://localhost:5000/securityRealm/addUser`, as showed in Table 6.5 .

Table 6.5: User registration in Jenkins

Option	Value
Username	`mike`
Password	`*********`
Full name	`Michele Sorcinelli`
E-mail address	`mikefender@cryptolab.net`

Then the access to unlogged user is disabled returning to the security configuration page and setting:

- Authorization: `Logged-in users can do anything`

6.4.3 Gasista Felice job configuration

Logging-in and navigating to the page `http://localhost:5000/newJob` a new job for the Gasista Felice project can be setted choosing the `Freestyle Project` option and `gasistafelice` as `Item name`. Configuration for the project is showed in Table 6.6 .

Table 6.6: Configuration for Gasista Felice Jenkins job (1)

Option	Value
Source Code Management	`Git`
Build	`Add build step -> Execute Shell`
Repository URL	`git@localhost:mike/gasistafelice.git`

More remote repositories and branches for the same project can be tracked, for example the upstream and developers forks can be tracked at the same time. Jenkins use the git client to check the hash of the commits introduced on pushes and avoid rebuilding of the same changes more times if found on different remote repositories or branches, for example when commits are merged from a developer fork to the upstream. Credentials for private repositories are required (Table 6.7).

Table 6.7: Credentials for Gasista Felice repository

Option	Value
Credentials	`Add -> SSH Username with private key`
Scope	`Global`
Username	`mike`
Private Key	content of private key body
Passphrase	passphrase chosen on key generation if exists

Then `mike` can be used for `Credentials` field. Additional project options are provided in Table 6.8 .

Table 6.8: Configuration for Gasista Felice Jenkins job (2)

Option	Value
Branches to build	`dev`, `master`
Build Triggers	`Poll SCM`
Schedule	

Then a shell script for the build has to be added from `Build -> Add a build step -> Execute shell`:

```
# override configuration
mv compose/ci.yml docker-compose.yml

# stop and rm old containers
sudo docker-compose stop
sudo docker-compose rm -f

# build images
sudo docker-compose build

# start containers
sudo docker-compose up -d

# prepare the database
sudo make dbtest

# run e2e tests
sudo docker-compose run --rm e2e

# stop and rm running containers
sudo docker-compose stop
sudo docker-compose rm -f
```

The `compose/ci.yml` is renamed to `docker-compose.yml`, replacing the configuration used for development with the one used in Continuous Integration builds, that is more similar to the production environment configuration. Docker Compose is used to build the container from the Dockerfiles, link and run them, then the test database is loaded and the tests are launched. Before finishing the build, the containers are stopped and removed. The reason for having `stop` and `rm` both at the top and bottom of the script is that if the build fails for some reason the containers are not stopped and removed. The build configuration can be tested clicking on the `Build Now` button in the project dashboard.

6.4.4 Remote repository hook configuration

In order to trigger the SCM poll from Jenkins when a push is performed (Figure 6.3), a hook for the `gasistafelice` repository in Gogs has to be added:

```
$ cd $HOME/gogs_data/git/repositories/mike/gasistafelice.git/hooks
# touch post-recieve
```

The `post-recieve` hook script content is:

```
#! /bin/sh
curl http://localhost:8080/git/notifyCommit\
?url=git@localhost:mike/gasistafelice.git \
2>/dev/null
```

Then the script ownership and execution permission have to be setted:

```
# chown 999:999 post-receive
# chmod +x post-receive
```

The configuration can be verified pushing to the remote `gasistafelice` repository:

```
$ cd path/to/gasistafelice/
$ git checkout origin/dev -b dev
Branch dev set up to track remote branch dev from origin.
Switched to a new branch 'dev'
$ git push gogs dev
Counting objects: 22, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (22/22), done.
Writing objects: 100% (22/22), 2.87 KiB | 0 bytes/s, done.
```

```
Total 22 (delta 13), reused 0 (delta 0)
remote: Scheduled polling of gasistafelice
To git@localhost:mike/gasistafelice.git
 * [new branch]      dev -> dev
```

As can be seen from the command output, a poll of `gasistafelice` has been scheduled by Jenkins and if changes are found on the `dev` or `master` branches, the project will be built. The tracked branches can be setted in the project configuration page. The status of all the project builds can be found at the url: `http://localhost:5000/job/gasistafelice/`.

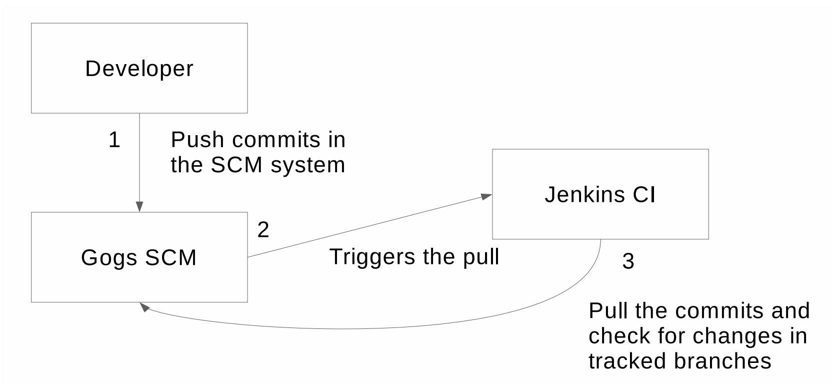

Developer
1
Push commits in the SCM system
Jenkins CI
2
Gogs SCM
Triggers the pull
3
Pull the commits and check for changes in tracked branches

Figure 6.3: Gogs triggers a SCM poll after a developer push

6.4.5 Scheduling of periodic polls and builds

The scheduling of periodic polls for the gasistafelice project can be configured from `http://localhost:5000/job/gasistafelice/configure` by filling the `Build Triggers -> Poll SCM -> Schedule` field of the form.

The syntax used for the scheduling is similar to the one used for *CRON* jobs. A nightly pool can be scheduled with this syntax:

```
0 0 * * *
```

This will execute a SCM poll every night at midnight, but a better approach is using the H tag:

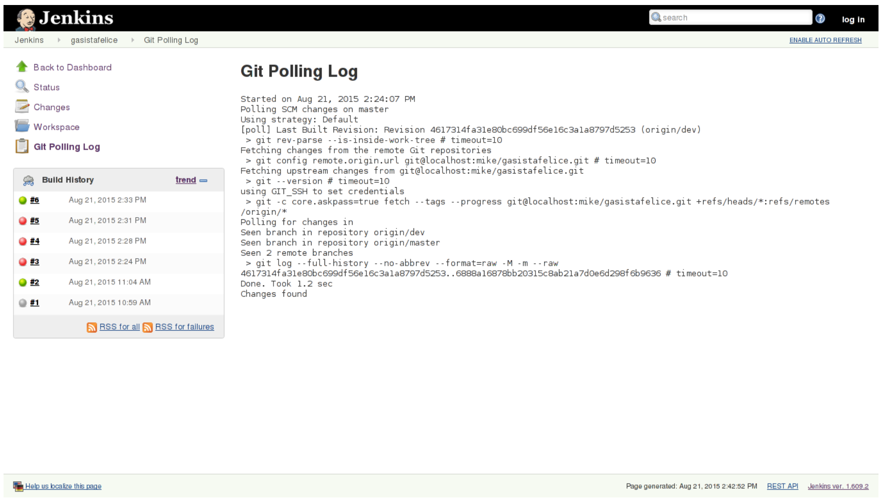
```
H H * * *
```

This syntax will assure the execution of the job once a day, but at a random chosen time, avoiding the overlapping of more jobs.

Periodical builds can be set filling the form in `Build Triggers -> Build Periodically` using the same syntax used for scheduling pools. For daily builds Docker Compose can be instructed to avoid the using of the Docker cache, ensuring that all the build is done from scratch, creating a new job with a modified build script where the build script:

```
...

# modified images build instruction
sudo docker-compose build --no-cache

...
```

Then the new job can be scheduled for daily or weekly execution. The pooling log (Figure 6.4) can be accessed from the `Git Pooling Log` button in the project Dashboard.



● #6	Aug 21, 2015 2:33 PM
● #5	Aug 21, 2015 2:31 PM
● #4	Aug 21, 2015 2:28 PM
● #3	Aug 21, 2015 2:24 PM
● #2	Aug 21, 2015 11:04 AM
● #1	Aug 21, 2015 10:59 AM

Git Polling Log

```
Started on Aug 21, 2015 2:24:07 PM
Polling SCM changes on master
Using strategy: Default
[poll] Last Built Revision: Revision 4617314fa31e80bc699df56e16c3a1a8797d5253 (origin/dev)
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repositories
 > git config remote.origin.url git@localhost:mike/gasistafelice.git # timeout=10
Fetching upstream changes from git@localhost:mike/gasistafelice.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git -c core.askpass=true fetch --tags --progress git@localhost:mike/gasistafelice.git +refs/heads/*:refs/remotes
/origin/*
Polling for changes in
Seen branch in repository origin/dev
Seen branch in repository origin/master
Seen 2 remote branches
 > git log --full-history --no-abbrev --format=raw -M -m --raw
4617314fa31e80bc699df56e16c3a1a8797d5253..6888a16878bb20315c8ab21a7d0e6d298f6b9636 # timeout=10
Done. Took 1.2 sec
Changes found
```

Page generated: Aug 21, 2015 2:42:52 PM

Figure 6.4: Jenkins Git pooling log

Parallel Jobs

Jenkins is configured by default to execute at most two jobs in parallel. Builds for the same project are never executed in parallel and are instead queued. To configure the maximum number of parallel builds, the system administrator can navigate to the Jenkins system configuration page and set the number of executors.

6.4.6 E-mail notifications

Jenkins can be configured to send e-mails with build reports. From the system configuration page, a STMP server and an email adress can be configured (if the `STMP server` field is left blank, `localhost` will be used).

A local STMP relay server can be installed and linked with Jenkins:

```
$ docker pull panubo/postfix
$ docker run --name postfix -d \
      -e MAILNAME='mail.example.org' \
      -e MYNETWORKS='127.0.0.1' \
      --net container:gogs \
      panubo/postfix
```

The recipient SMTP server can be configured with a anti-spam filter that can block this kind of e-mail notifications, especially if the IP address is domestic or dynamic:

```
postfix/smtp[200]: B5F3581CEE: to=<jenkins@mail.example.org>,
relay=mail.befair.it[80.85.85.154]:25, delay=0.58,
delays=0.02/0/0.48/0.08, dsn=5.7.1,
status=bounced (host mail.befair.it[80.85.85.154]
said: 554 5.7.1 Service unavailable;
Client host [82.51.4.196] blocked using zen.spamhaus.org;
http://www.spamhaus.org/query/bl?ip=82.51.4.196
(in reply to RCPT TO command))
```

To work around this problem, a different IP can be used or a remote SMTP server can be configured. An e-mail notification for a project can be added from the job configuration page:

- Add post-build action -> E-mail Notification
- Recipients: whitespace-separated list of recipient addresses

The available conditions are:

- Send e-mail for every unstable build
- Send separate e-mails to individuals who broke the build

6.4.7 Continuous delivery

In order to provide Continuous Delivery Jenkins a new job can be added and configured to act as a post build hook. This job will execute the trigger for Continuous Delivery, that theoretically consists in accessing an API provided by the production stack to deploy the update.

After this new job is configured, it can be triggered as a post-build action of the job used to build and test the involved software. The post-build action can be configured from the job configuration page:

- Add post-build action -> Build other projects
- Projects to Build -> The name of the post-build hook job

The available conditions are:

- Trigger only if build is stable
- Trigger only if the build is unstable
- Trigger even if the build fails

Chapter 7

Conclusions

This thesis describes the organization of a web application in a container based infrastructure, and the configuration of development, testing and Continuous Integration environments for that application.

The configuration of the development environment containers is an added value for the team because permits its developers to work in a standardized, reproducible, and easy to obtain environment, allowing them to never worry again of installation procedures and operating systems related problems, and thus increasing their productivity. This benefits act also as an incentive to external collaborations, because the effort to start the development is reduced.

The configuration of the end-to-end test environment containers lays the foundation for developers writing end-to-end tests that can be executed without additional configurations, and those tests can be used to reproduce reported bugs and verify their resolution, as well as specifications of new features to implement.

The implemented end-to-end tests give more value to the software product because they verify the correctness of the main features of the application user interface, can be used as demonstration for the customer, and give security to developers when applying changes to code covered by automated tests.

The implemented Continuous Integration solution increases the productivity of the entire team, giving rapid feedbacks for every change introduced in the software, and testing the application in a context similar to the production environment.

All the implemented solutions run in containerized infrastructures, so this thesis demonstrates how working with containers can be easy and productive.

The future developments regard the reorganization of the Gasista Felice application to a microservices architecture, that can be easily obtained thanks to the migration to a container based infrastructure, and the extension of the implemented Continuous Integration system to provide integration with git [1] server advanced features (such as the automatic testing of the incoming pull requests [23]), to improve even more the productivity of the development team.

The implemented Continuous Integration system can be integrated with the production stack in order to provide Continuous Delivery, and this will be done soon as the production stack is predisposed to the container infrastructure.

Most of the work presented in this thesis is flexible and can be easily reused in the development of other applications.

Appendix A

Source Code

This appendix includes most source code files relevant to the thesis:

- Dockerfiles
- Docker Compose files
- End-to-end Javascript files
- Makefile
- Bash functions for the CI system

The source code of Gasista Felice project, including the backend and frontend is available in `master-dj17` branch of the `befair/gasistafelice` GitHub repository: `https://github.com/befair/gasistafelice/`. The code is updated at this revision:

```
commit ff02193cee44a03772595450faf419b5f7cdbdb8
Merge: 8e59faf 6cdf4df
Author: Luca Ferroni <luca@befair.it>
Date:   Thu Sep 3 12:09:40 2015 +0200

    Merge pull request #151 from michelesr/dev

    E2E: improved test/e2e/spec.js
```

A.1 Dockerfiles

Gasista Felice application server

```
FROM kobe25/uwsgi-python2:latest

MAINTAINER Antonio Esposito "kobe@befair.it"
```

```
ENV LC_ALL                      it_IT.UTF-8
ENV LANG                        it_IT.UTF-8
ENV LANGUAGE                    it_IT.UTF-8

# the following ENV directives have been reformatted
# for printing purposes, the correct syntax is
# "ENV VAR_NAME value", in one line

ENV PYTHONPATH
    /code:/code/gasistafelice:/usr/local/lib/python2.7/site-packages
ENV UWSGI_CHDIR
    /code/gasistafelice
ENV UWSGI_WSGI_FILE
    /code/gasistafelice/gf/wsgi.py
ENV DJANGO_SETTINGS_MODULE
    gf.settings
ENV UWSGI_STATIC_MAP
    /static=/code/gasistafelice/static
ENV UWSGI_STATIC_SAFE
    /usr/local/lib/python2.7/site-packages/django/contrib/admin/static/admin

COPY deps/debian /code/gasistafelice/deps/debian
RUN apt update && \
    apt install -y $(cat /code/gasistafelice/deps/debian) && \
    rm -rf /var/lib/apt/lists/*

COPY deps/locale.gen /etc/locale.gen
RUN locale-gen

COPY deps/ /code/gasistafelice/deps/
RUN pip install -r /code/gasistafelice/deps/dev.txt

COPY ./ /code/gasistafelice/
WORKDIR /code/gasistafelice/
```

The Dockerfile for `kobe25/uwsgi-python2:latest` is:

```
FROM python:2.7.7

MAINTAINER Antonio Esposito "kobe@befair.it"

ENV DEBIAN_FRONTEND noninteractive
ENV PYTHONUNBUFFERED 1
ENV PYTHONPATH /code:/usr/local/lib/python2.7/site-packages
ENV UWSGI_CALLABLE app
ENV UWSGI_PYTHON_AUTORELOAD 1 #
ENV UWSGI_PY_TRACEBACKER
ENV UWSGI_MASTER true
ENV UWSGI_MASTER_AS_ROOT true
ENV UWSGI_UID app
ENV UWSGI_GID app
ENV UWSGI_UWSGI_SOCKET 0.0.0.0:5000
ENV UWSGI_NO_ORPHANS true
ENV UWSGI_VACUUM true
ENV UWSGI_LOG_DATE true
ENV UWSGI_LAZY_APPS false
ENV UWSGI_WORKERS 2
ENV UWSGI_THREADS 1
ENV UWSGI_ENABLE_THREADS true
ENV UWSGI_BUFFER_SIZE 65536
ENV UWSGI_MAX_REQUESTS 128
ENV UWSGI_HARAKIRI 120
ENV UWSGI_HARAKIRI_VERBOSE true
ENV UWSGI_THUNDER_LOCK true
ENV PGDATABASE app
ENV PGUSER app
ENV PGPASSWORD app
ENV PGHOST db
ENV PGPORT 5432

RUN groupadd -r app && \
    useradd -r -g app -d /code app
RUN apt update && \
    apt install -y \
      build-essential \
      python-dev \
      python-setuptools && \
    rm -rf /var/lib/apt/lists/*
RUN pip install \
      'uWSGI >=2.0, <2.1'
EXPOSE 5000 CMD ["uwsgi"]
```

Gasista Felice proxy server

```
FROM kobe25/nginx:latest

MAINTAINER Antonio Esposito "kobe@befair.it"
```

```
COPY site.conf /etc/nginx/conf.d/site.conf
```

The `site.conf` file is:

```
server {
  listen 8080 default_server;
  #listen 8443 ssl spdy default_server;

  server_name _;
  root /code;

  charset utf-8;
  client_max_body_size 75M;
  client_body_timeout 600s;
  #keepalive_timeout 5;

  location = /favicon.ico {
    log_not_found off;
    access_log off;
  }

  location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
  }

  location ~ /\. {
    deny all;
    log_not_found off;
    access_log off;
  }

  location = / {
    proxy_pass      http://front:5000/ui/index.html;
    expires max;
  }

  location /components/ {
    proxy_pass      http://front:5000/ui/components/;
    expires max;
  }
  location /ui/bower_components/ {
    proxy_pass      http://front:5000/libs/bower_components/;
    expires max;
  }

  location /ui/ {
    proxy_pass      http://front:5000;
    expires max;
  }

  location /static/ {
    include          uwsgi_params;
    uwsgi_pass       uwsgi://back:5000;
    expires max;
  }

  location /media/ {
    include          uwsgi_params;
    uwsgi_pass       uwsgi://back:5000;
  }

  location /api/ {
    gzip             on;
    gzip_types       application/json;
    gzip_min_length  1000;

    include          uwsgi_params;
    uwsgi_pass       uwsgi://back:5000;
  }

  location /gasistafelice/ {
    include          uwsgi_params;
    uwsgi_pass       uwsgi://back:5000;
  }

  location / {
    return 404;
  }
}
```

The Dockerfile for `kobe25/nginx:latest` is:

```
FROM nginx:1.9
```

```
MAINTAINER Antonio Esposito "kobe@befair.it"

RUN rm -rf /etc/nginx/conf.d/*

COPY nginx.conf /etc/nginx/nginx.conf
```

Gasista Felice frontend server

```
FROM iojs:2.5

MAINTAINER Antonio Esposito "kobe@befair.it"

RUN npm config set registry http://registry.npmjs.org/

COPY deps/npm /code/ui/deps/npm
RUN npm install -g $(cat /code/ui/deps/npm)

COPY ./bower.json /code/libs/bower.json
RUN cd /code/libs/ && bower install --allow-root

EXPOSE 5000

COPY ./ /code/ui/
WORKDIR /code/ui/

CMD ["harp", "server", "-i", "0.0.0.0", "-p", "5000", "/code"]
```

End-to-end test framework

```
FROM michelesr/protractor:latest

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

COPY . /code
```

The Dockerfile for `michelesr/protractor:latest` is:

```
FROM iojs:3

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

RUN npm install -g protractor

RUN mkdir /code
```

```
WORKDIR /code

CMD ["protractor", "conf.js"]
```

Jenkins Dockerfile

```
FROM jenkins:latest

MAINTAINER Michele Sorcinelli "mikefender@cryptolab.net"

USER root
RUN apt-get update \
      && apt-get install -y python python-pip sudo \
      && rm -rf /var/lib/apt/lists/*
RUN pip install 'docker-compose==1.4'
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers

USER jenkins
COPY plugins.txt /usr/share/jenkins/plugins.txt
RUN /usr/local/bin/plugins.sh /usr/share/jenkins/plugins.txt
```

The `plugins.txt` file is:

```
scm-api:latest
git-client:latest
git:latest
greenballs:latest
```

A.2 Docker Compose files

Gasista Felice

```
proxy:
  image: befair/gasistafelice-proxy:latest
  volumes:
    - ./proxy/site.conf.dev:/etc/nginx/conf.d/site.conf:ro
  ports:
    - '127.0.0.1:8080:8080'
    - '127.0.0.1:8443:8443'
  links:
    - front
    - back

front:
  image: befair/gasistafelice-front:latest
  volumes:
    - ./ui:/code/ui:rw

back:
  image: befair/gasistafelice-back:latest
  volumes:
    - ./gasistafelice:/code/gasistafelice:ro
    - ./gasistafelice/fixtures:/code/gasistafelice/fixtures:rw
    - /tmp/gf_tracebacker:/tmp/tracebacker:rw
    - /tmp/gf_profiling:/tmp/profiling:rw
  ports:
    - '127.0.0.1:7000:7000'
  links:
    - db
  env_file: ./compose/settings.env

db:
  image: postgres:9.4
  env_file: ./compose/settings.env
```

The compose/settings.env file is:

```
APP_ENV=dev
POSTGRES_USER=app
POSTGRES_PASSWORD=app
UWSGI_UID=root
UWSGI_GID=root
UWSGI_HTTP=0.0.0.0:7000
UWSGI_WORKERS=1
UWSGI_PY_TRACEBACKER=/tmp/tracebacker
```

Testing

```
hub:
  image: selenium/hub:latest

firefox:
  image: selenium/node-firefox-debug:latest
  links:
    - hub
    - proxy
  ports:
    - '127.0.0.1:5900:5900'
  env_file:
    - ./test/e2e/settings.env
```

```
chrome:
  image: selenium/node-chrome-debug:latest
  links:
    - hub
    - proxy
  ports:
    - '127.0.0.1:5901:5900'
  env_file:
    - ./test/e2e/settings.env

e2e:
  image: michelesr/protractor:latest
  volumes:
    - ./test/e2e:/code:ro
  links:
    - hub
```

The `test/e2e/settings.env` file is:

```
SCREEN_WIDTH=1920
SCREEN_HEIGHT=1080
```

CI

```
proxy:
  build: ./proxy
  ports:
    - '127.0.0.1:8080:8080'
    - '127.0.0.1:8443:8443'
  links:
    - front
    - back

front:
  build: ./ui

back:
  build: ./gasistafelice
  links:
    - db
  env_file: ./compose/settings_ci.env

db:
  image: postgres:9.4
  env_file: ./compose/settings_ci.env

hub:
  image: selenium/hub:latest

e2e:
  build: ./test/e2e
  links:
    - hub

firefox:
  image: selenium/node-firefox-debug:latest
  links:
    - hub
    - proxy
  env_file:
    - ./test/e2e/settings.env

chrome:
  image: selenium/node-chrome-debug:latest
  links:
    - hub
    - proxy
  env_file:
    - ./test/e2e/settings.env
```

The `compose/settings_ci.env` file is:

```
APP_ENV=prod
POSTGRES_USER=app
POSTGRES_PASSWORD=app
```

A.3 End-to-end Javascript files

Protractor configuration file

```
// conf.js
exports.config = {
  seleniumAddress: 'http://hub:4444/wd/hub',
  specs: ['spec.js'],
  multiCapabilities: [
    { browserName: "firefox" },
    { browserName: "chrome" }
  ]
}
```

Test routine specification file

```
var handleAlertPopup = function() {
  var EC = protractor.ExpectedConditions;
  browser.wait(EC.alertIsPresent(), 5000);
  browser.switchTo().alert().accept();
};

describe('GF ng-app', function() {

  [0,1].map(function(index) {

    it('should have a title', function() {
      if (!index)
        browser.driver.manage().window().maximize();
      else
        browser.driver.manage().window().setSize(768, 1024);
      browser.get('http://proxy:8080/');
      expect(browser.getTitle()).toEqual('Gasista Felice');
    });

    it('should connect to the user order page', function() {
      // fill login form
      element(by.model('app.username')).sendKeys('01gas1');
      element(by.model('app.password')).sendKeys('des');

      // click on 'GO' button!
      // $$(selector) = element(by.css(selector))
      $$('#go').click();

      // check current url
      expect(browser.getLocationAbsUrl())
        .toBe('/order/');

      // check user displayed name
      if (index) {
        $$('.navbar-toggle').click();
        expect(element(by.binding('person.display_name')).getText())
          .toBe("Gasista_01 DelGas_01");
        $$('.navbar-toggle').click();
      }
      else
        expect(element(by.binding('person.display_name')).getText())
          .toBe("Gasista_01 DelGas_01");
```

```
});

it('should increment/decrement the price/qty when "+/-" are clicked',
   function () {
  // get the second item in the table
  var item = element.all(
              by.repeater('product in order.pm.products')
            ).get(1);

  // click 20 time on '+'
  for (var i=0; i < 20; i++)
    item.$$('.glyphicon-plus').click();

  // click 10 times on '-'
  for (var i=0; i < 10; i++)
    item.$$('.glyphicon-minus').click();

  // qty should be 10
  expect(item.element(by.model('product.quantity'))
    .getAttribute('value'))
      .toBe('10');

  // price should be 250 euros
  expect(item.element(by.binding('product.quantity'))
    .getText())
      .toBe('€ 250,00');
});

it('should never decrement the price/qty under 0',
   function () {
  var item = element.all(
              by.repeater('product in order.pm.products')
            ).get(1);

  for (var i=0; i < 20; i++)
    item.$$('.glyphicon-minus').click();

  expect(item.element(by.model('product.quantity'))
    .getAttribute('value'))
      .toBe('0');

  expect(item.element(by.binding('product.quantity'))
    .getText())
      .toBe('€ 0,00');
```

```
});

it('should add a product to the basket', function() {
  var item = element.all(
               by.repeater('product in order.pm.products')
             ).get(2);

  // set the quantity to 3
  item.element(by.model('product.quantity')).clear();
  item.element(by.model('product.quantity')).sendKeys('3');

  // add to the basket
  element(by.buttonText('Aggiungi al paniere')).click();

  // handle the alert popup
  handleAlertPopup();

  // go to the basket
  browser.setLocation('basket');

  // get the first order
  item = element.all(
           by.repeater('item in basket.open_ordered_products')
         ).get(0);

  // get all the column from the first order
  var columns = item.$$('td');

  // expects to have 8 columns (counting the hidden ones)
  expect(columns.count())
    .toBe(8);

  // check the fields
  expect(columns.get(0)
    .getText())
      .toBe('Ord. 59');
  expect(columns.get(1)
    .getText())
      .toBe('Fornitore 01');
  expect(columns.get(2)
    .getText())
      .toBe('Scarpe Uomo (paio)');
  expect(columns.get(3)
    .getText())
```

```
          .toBe('€ 20,00');
        expect(item.element(by.model('item.quantity'))
          .getAttribute('value'))
            .toBe('3');
        expect(columns.get(6)
          .getText())
            .toBe('€ 60,00');
    });

    it('should logout', function() {
      if(index)
        $$('.navbar-toggle').click();
      $$('#btn-logout').click();
    });
  });
});
```

A.4 Makefile

```
help:
	@echo 'make              Print this help'
	@echo
	@echo 'Whole app commands:'
	@echo 'make up           Download and start all'
	@echo 'make ps           Container status'
	@echo 'make logs         See all logs'
	@echo 'make stop         Stop all containers'
	@echo 'make restart      Restart all containers'
	@echo 'make rm           Delete containers'
	@echo 'make test         Run all tests'
	@echo
	@echo 'Container commands:'
	@echo 'make logs-back    See only backend logs'
	@echo 'make back         Debug in backend via iPython'

test-cat.yml: docker-compose.yml compose/test.yml Makefile
	@cat docker-compose.yml compose/test.yml > test-cat.yml

clean:
	@rm test-cat.yml

up:
	@docker-compose up -d
	@docker-compose ps

logs log:
	@docker-compose logs

logs-back log-back:
	@docker-compose logs back

start:
	@docker-compose start
	@docker-compose ps

stop: test-cat.yml
	@docker-compose -f test-cat.yml stop
	@docker-compose ps

restart:
	@docker-compose restart
	@docker-compose ps

ps:
	@docker-compose ps

t:
	@docker-compose run --rm test /bin/bash

front fe frontend ui:
	@docker-compose run --rm front /bin/bash

back be backend api:
	@docker-compose run --rm back /bin/bash

shell:
	@docker-compose run --rm back django-admin shell

dbshell:
	@docker-compose run --rm back django-admin dbshell

dbinit:
	@docker-compose run --rm back django-admin makemigrations --noinput
	@docker-compose run --rm back django-admin migrate
	@docker-compose run --rm back django-admin init_superuser

dbtest: dbclean
```

```
    @docker-compose run --rm back psql \
    -f /code/gasistafelice/fixtures/test.sql

dbdump:
    @docker-compose run --rm back pg_dump \
    -f /code/gasistafelice/fixtures/test.sql app

dbclean:
    @docker-compose run --rm back dropdb app
    @docker-compose run --rm back createdb app -O app

rm: stop
    @docker-compose -f test-cat.yml rm -v -f

rmall: rm
    @docker rmi -f befair/gasistafelice-{front,back}

rmc:
    @docker rm -f $(docker ps -aq)

rmi: rmc
    @docker rmi -f $(docker images -aq)

test: test-info test-unit test-integration test-e2e
    @echo 'All tests passed!'

test-info:
    @echo 'To prepare the test db (this will clear your data):'
    @echo '    $$ make dbtest'
    @echo

test-unit:
    @echo 'Unit test: not implemented yet'

test-integration:
    @echo 'Integration test: not implemented yet'

test-e2e: test-cat.yml
    @echo 'End-to-end test: running protractor'
    @docker-compose -f test-cat.yml up -d
    @sleep 5
    @docker-compose -f test-cat.yml run --rm e2e
```

A.5 Bash functions for the CI system

```
gogs() {
  docker run --name gogs -d \
             -p 3000:3000 \
             -p 32:22 \
             -p 5000:8080 \
             -v $HOME/gogs_data:/data \
             gogs/gogs
}

jenkins() {
  docker run --name jenkins -d \
         -v /var/run/docker.sock:/var/run/docker.sock \
         -v /usr/lib/libdevmapper.so.1.02:/usr/lib/libdevmapper.so.1.02 \
         -v $(which docker):/usr/bin/docker \
         -v $HOME/jenkins_data:/var/jenkins_home \
         --net container:gogs \
         michelesr/jenkins
}


postfix() {
  docker run --name postfix -d \
         -e MAILNAME='micheles.no-ip.org' \
         -e MYNETWORKS='127.0.0.1' \
         --net container:gogs \
         panubo/postfix
}
```

Bibliography

[1] *Git*, Available: https://www.git-scm.com/

[2] *Protractor - end to end testing for AngularJS*, Available: https://angular.github.io/protractor/

[3] beFair development team, *Gasista Felice*, Available: http://www.gasistafelice.org/

[4] Benita Matofska, "What is the Sharing Economy?", *The People Who Share*, Available: http://www.thepeoplewhoshare.com/blog/what-is-the-sharing-economy

[5] Docker Inc. © 2015, *Docker - Build, Ship, and Run Any App, Anywhere*, Available: https://www.docker.com/

[6] Docker Inc. © 2015, "Supported Installation", *Docker Documentation*, Available: https://docs.docker.com/installation/

[7] Docker Inc. © 2015, *Docker Hub Registry - Repositories of Docker Images*, Available: https://hub.docker.com/

[8] Docker Inc. © 2015, *Docker Compose*, Available: https://www.docker.com/docker-compose

[9] Docker Inc. © 2015, *Introducing Docker Content Trust*, Available: https://blog.docker.com/2015/08/content-trust-docker-1-8/

[10] Docker Inc. © 2015, *Docker Toolbox*, Available: https://www.docker.com/toolbox

[11] Don Wells © 1997-1999, All rights reserved, "Integrate Often", *Extreme Programming*, Available: http://www.extremeprogramming.org/rules/integrateoften.html

[12] Eric Steven Raymond © 2013, CC-BY-ND 1.0, "Chapter 1 - Philosophy", *Basics of the Unix Philosophy*, Available: http://www.faqs.org/docs/artu/ch01s06.html

[13] Free Software Foundation Inc. © 2015, CC-BY-ND 4.0, "What is free software?" *GNU Project - Free Software Foundation* Available: https://www.gnu.org/philosophy/free-sw.html

[14] Galina Pankratova, InfoWorld Inc. © 1994-2015, All rights reserved, 4 December 2014, *Why io.js decided to fork Node.js*, Available: http://www.infoworld.com/article/2855057/application-development/why-iojs-decided-to-fork-nodejs.html

[15] Google Inc. © 2010-2015, CC-BY 3.0, *AngularJS - Superheroic JavaScript MVW Framework*, Available: https://angularjs.org/

[16] Google Inc. © 2010-2015, CC-BY 3.0, "Step 4: Two-Way Data Binding", *AngularJS Tutorial*, Available: https://docs.angularjs.org/tutorial/step_04

[17] Google Inc. © 2010-2015, CC-BY 3.0, "ngModel directive", *AngularJS API Reference*, Available: https://docs.angularjs.org/api/ng/directive/ngModel

[18] Google Inc. © 2010-2015, CC-BY 3.0, "ngBind directive", *AngularJS API Reference*, Available: https://docs.angularjs.org/api/ng/directive/ngBind

[19] Google Inc. © 2010-2015, CC-BY 3.0, "ngRepeat directive", *AngularJS API Reference*, Available: https://docs.angularjs.org/api/ng/directive/ngRepeat

[20] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, © 2001, *Manifesto for Agile Software Development*, Available: http://www.agilemanifesto.org/

[21] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, © 2001, *Principles behind the Agile Manifesto*, Available: http://www.agilemanifesto.org/principles.html

[22] Nick Sullivan, Cloudflare Inc. © 2015, 10 March 2014, "ECDSA vs RSA", *ECDSA: The digital signature algorithm of a better internet*, Available: https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/

[23] Mark Johnson, © 2007-2014 University of Oxford, CC-BY-SA, 08 November 2013, "What is a Pull Request", *OSS Watch*, Available: http://oss-watch.ac.uk/resources/pullrequest

[24] Martin Fowler ©, 01 May 2006, *Continuous Integration*, Available: http://www.martinfowler.com/articles/continuousIntegration.html

[25] Martin Fowler ©, 10 March 2014, *Microservices*, Available: http://martinfowler.com/articles/microservices.html

[26] Mike Loukides, O'Reilly Media, Inc. © 2015, June 7 2012 "What is DevOps?", *the agile admin*. Available: http://theagileadmin.com/what-is-devops/

[27] Mozilla Developer Network and individual Contributors © 2005-2015, CC-BY-SA 2.5, "Array.prototype.map()", *Web Technology for developers*, Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

[28] Wikimedia Foundation Inc. and Contributors ©, CC-BY-SA 3.0, last modified on 18 August 2012, "Docker (software)", *Wikipedia, The Free Encyclopedia*, Available: https://en.wikipedia.org/wiki/Docker_(software)

Acknowledgments

First and foremost I would like to thank my parents, that supported me through all my educational career, providing the necessary resources and encouragement. Without their support, I would never been able to achieve these results. I would like also to give my gratitude to Luca Ferroni, that provided a great support to this work, welcomed me with open arms in his organization, and, most important, he trusted in my ability even more than I did.