Anunci.us

Protocolos, tecnologías y servicios de Internet

Subject code: 51336

Sergio Anguita Lorenzo

Project requirements

According to student guide, the project must fulfill next guidelines:

> *TRABAJO INVIDIDUAL:*
>
> *Desarrollo de una solución basada en Internet con parte tanto cliente como servidora por el estudiante aplicando los protocolos y tecnologías de Internet y Web, de última generación, estudiadas. Se documentará el uso de las herramientas aplicadas. Se contempla una entrega final, obligatoria y evaluable. El trabajo será evaluado sobre 7 puntos. La documentación asociada y la presentación del proyecto se realizarán en inglés.*

Project description

The aim of this project is to create a RIA (Rich Internet Application) hosted on a cloud service. The selected RIA will focus on the concept of online shop in where anyone can sell and buy items. We could take as a reference **milanuncios.com.** So, taking a brief look to this platform, we are going to build something similar. With this idea in mind and in order to make it better (at least responsive) we will have a better platform than many others, at least from design point of view.

In the next section, project functional and non functional requirements are specified.

Functional requirements

In this section functional requirements are summarized.

- Application must have push notifications
- User login must be done using Single Sign On system
- Logged user will have the ability to edit and delete their published items.
- Not logged users can only publish an item
- Not logged users will not be able to edit or delete their items until the login.
- User defines the category where the item belongs to.

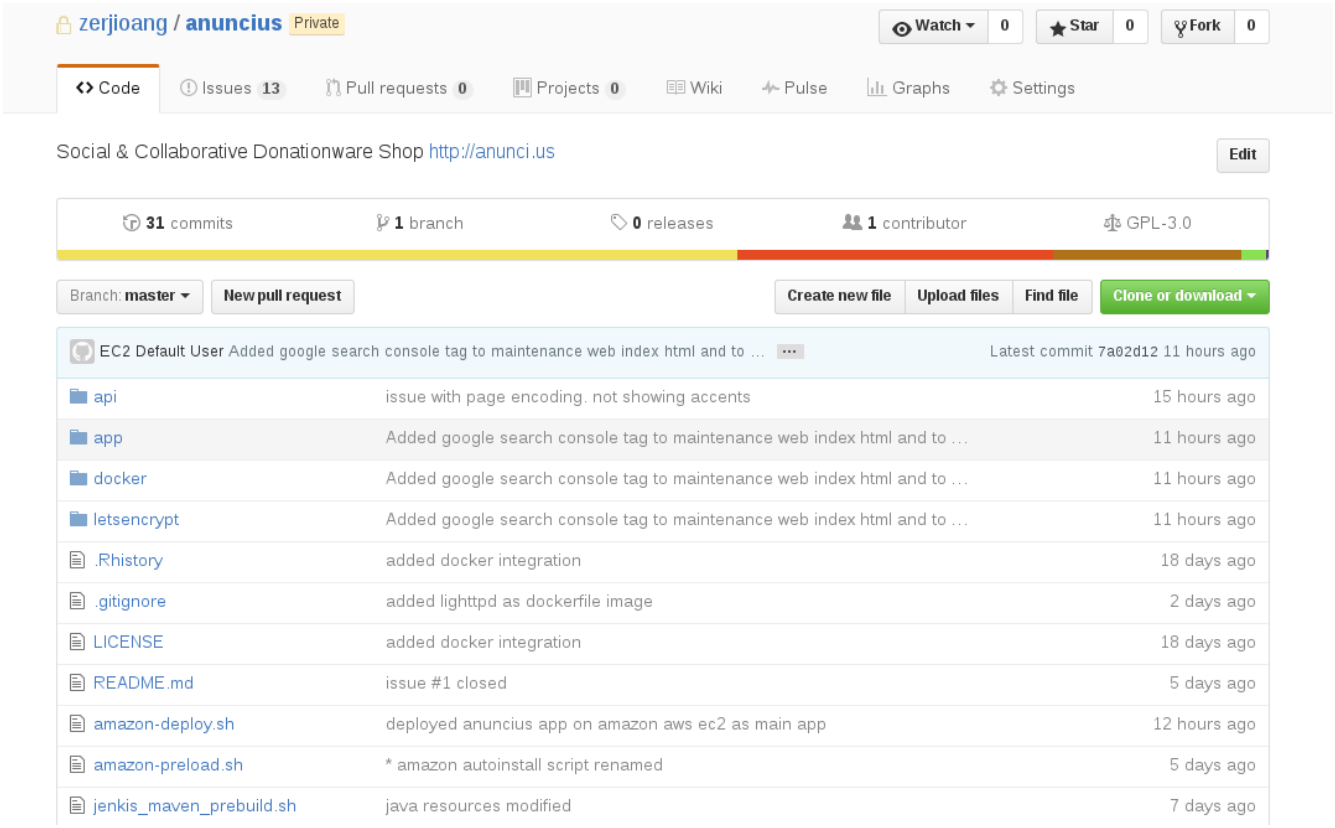
Non-functional requirements

In this section non functional requirements are summarized.

- Application must have a responsive design.

Project management

Project management is done using already known **Github** platform for code hosting and issue management. Current source code is located under a private repository. Repository URL is given below.



https://github.com/zerjioang/anuncius

Architecture

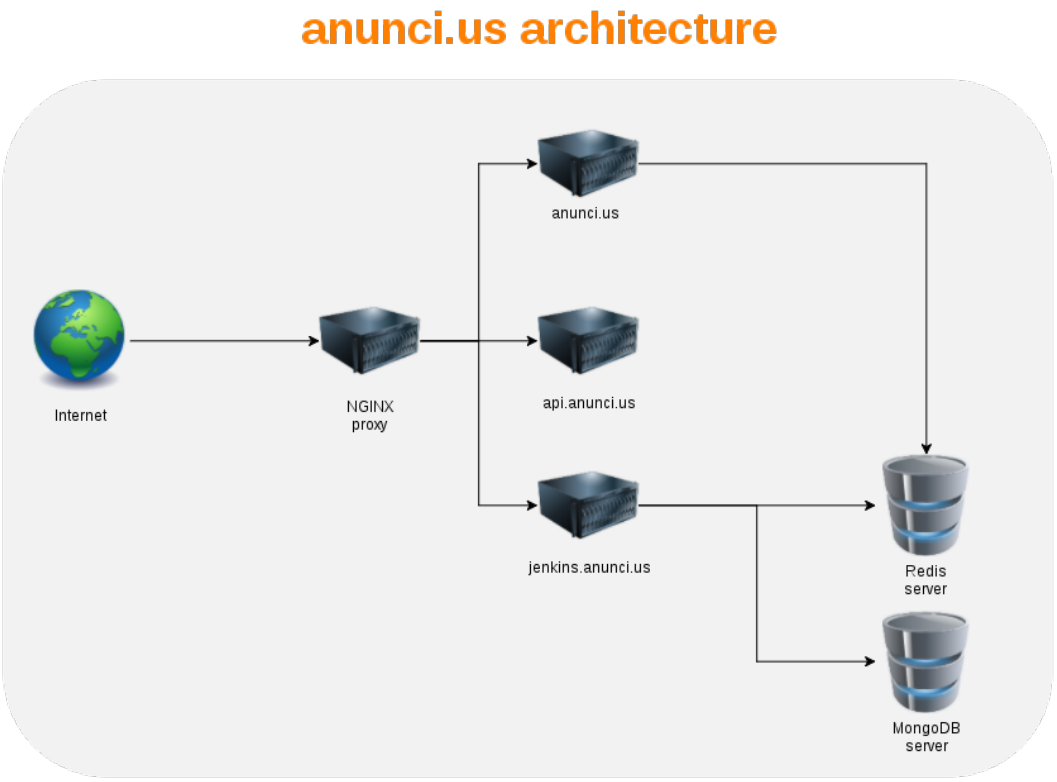
In this chapter, platform architecture is described. A different aproach is taken for architecture, since all platform will be based on Docker containers. For those of you, who do not know what a docker container is, let me show you:

> *Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.*

Using Docker as main environment for development it gives some advantages:

- More secure environment caused by shared docker env.
- Deploy and scale easily because docker containers can run almost everywhere: laptop, dedicated server, virtual machine, etc.
- Superfast to move from one environment to other. Easy migrations.

Interal design



Microservices

The platform is built on top of a series of services that run on a shared environment on the same machine. This services are:

- A Redis server for cache management
- A Mongo database for NoSQL data storage
- A tomcat webserver for application hosting
- A jenkins application for CI development.
- A Nginx based server as reverse proxy for routing mainly

Why use a reverse proxy with Docker

A reverse proxy server is a server that typically sits in front of other web servers in order to provide additional functionality that the web servers may not provide themselves.

For example, a reverse proxy can provide SSL termination, load balancing, request routing, caching, compression or even A/B testing.

When running web services in docker containers, it can be useful to run a reverse proxy in front of the containers to simplify deployment.

Docker containers are assigned random IPs and ports which makes addressing them much more complicated from a client perspective. By default, the IPs and ports are private to the host and cannot be accessed externally unless they are bound to the host.

Binding the container to the hosts port can prevent multiple containers from running on the same host. For example, only one container can bind to port 80 at a time. This also complicates rolling out new versions of the container without downtime since the old container must be stopped before the new one is started.

A reverse proxy can help with these issues as well as improve availability by facilitating zero-downtime deployments.

Development

Google Oauth login implementation

Following the official guidelines of Google (https://developers.google.com/identity/sign-in/web/devconsole-project) an Oauth login based solution has been implemented for Single Sign On management. Users log into the platform using their Google accounts. For this process, the only requirements, are to include in html code next tag.

```
<meta name="google-signin-client_id" content="119996567226-
d25nbob8g6gahivnaevimvnpmbjoqqdi.apps.googleusercontent.com">
```

Including this tag and adding the proper Google js file, Oauth login will be done.

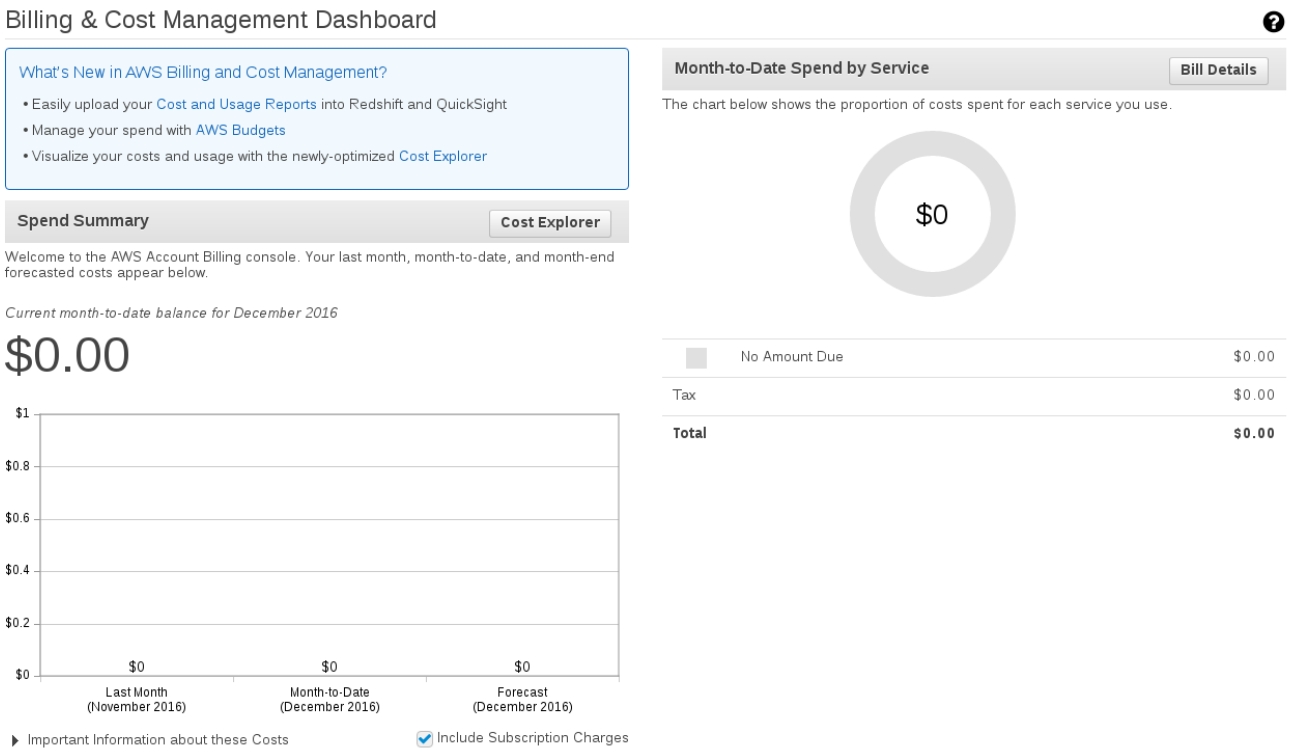
It is also required to declare where the button is going to be injected and which callback functions are going to trigger.

All this process is handled by next piece of code:

```
function onSuccess(googleUser) {
    console.log('Logged in as: ' + googleUser.getBasicProfile().getName());
    onSignIn(googleUser);
}

function onFailure(error) {
  console.log(error);
}

function renderButton() {
  gapi.signin2.render('my-signin2', {
    'scope': 'profile email',
    'width': 300,
    'height': 42,
    'longtitle': true,
    'theme': 'dark',
    'onsuccess': onSuccess,
    'onfailure': onFailure
  });
}

function onSignIn(googleUser) {
    var profile = googleUser.getBasicProfile();
    console.log('ID: ' + profile.getId());
    console.log('Name: ' + profile.getName());
    console.log('Image URL: ' + profile.getImageUrl());
    console.log('Email: ' + profile.getEmail());
    return profile;
  }
function signOut() {
    var auth2 = gapi.auth2.getAuthInstance();
    auth2.signOut().then(function () {
        console.log('User signed out.');
    });
}
```

Deployment

Application deployment is done on **AWS (Amazon Web Services)** using a t.micro style **EC2** instance thanks to Amazon Free-Tier. Thanks to this tier, billing cost are **$0.00** as can be seen on

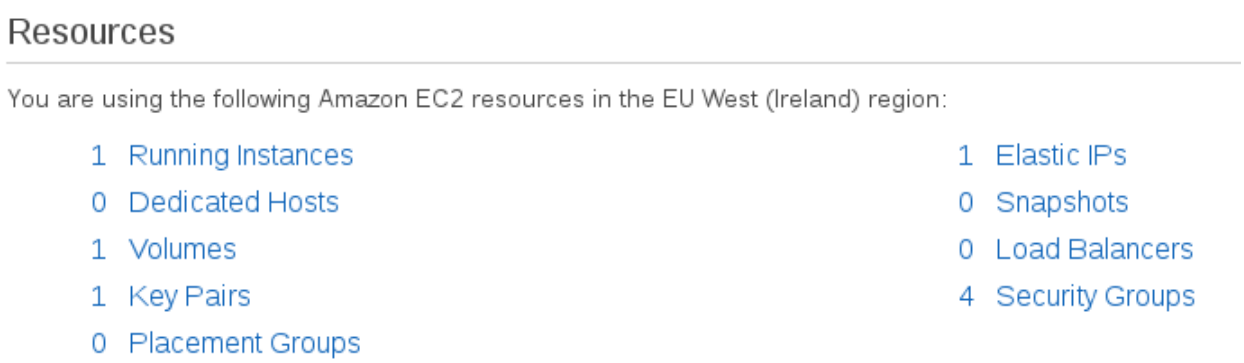


https://console.aws.amazon.com/billing/home?region=eu-west-1#/

Before uploading the application, there are several thing that have to be done in order to properly configure the server.

Used resources

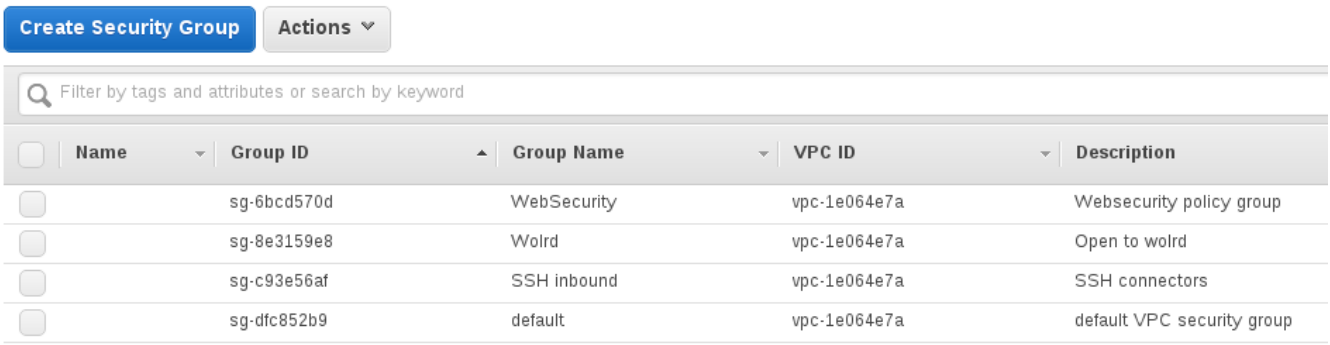
Platform used resources by Amazon EC2



Server hardening

In this section, how server is secured for unwanted access is described. Current security measurements are based on AWS security groups for limiting server access. Current configuration is shown:

AWS Security Groups



Name	Group ID	Group Name	VPC ID	Description
	sg-6bcd570d	WebSecurity	vpc-1e064e7a	Websecurity policy group
	sg-8e3159e8	Wolrd	vpc-1e064e7a	Open to wolrd
	sg-c93e56af	SSH inbound	vpc-1e064e7a	SSH connectors
	sg-dfc852b9	default	vpc-1e064e7a	default VPC security group

Used security groups are:

- **WebSecurity**: allows ANY incoming connection on ports 80 and 443 using HTTP and HTTPS protocols
- **Wolrd**: allows ANY incoming connection on ANY port using ANY protocol
- **SSH inbounds**: allows SSH connections via TCP on port 22 from my IP range.
- **Default**: allows EVERYTHING

With this security groups in mind, server instance has linked to SSH inbounds group and to WebSecurity group in order to have it available as Webserver and available for ssh connections.

DNS configuration

For DNS configuration, it is recommended to use Amazon Route 53 service, but since this service is out of the Amazon Free-tier, an alternative solution is found using an Elastic IP.

Configuring a DNS for our instance is as simple as generating an Elastic IP and linking to our domain using a DNS A record. Once this is done, the server will be available via http://anunci.us and http://www.anunci.us

Take into account that Amazon will charge you if you get an Elastic IP and you do not use it with ANY instance. Otherwise, feel free to use it on your t.micro instance.

Getting Elastic IP

Elastis Ips are like static IP and do not change when you stop or terminate your instance, so we can use if for DNS redirection. This Ips are available on Network & Security > Elastic Ips Section.

As you can see on the image above, my Elastic IP is 52.212.54.13 and this is the one that will be linked to DNS A Record.

Registering the IP as DNS A

> An A record maps a domain name to the IP address (IPv4) of the computer hosting the domain. Simply put, an A record is used to find the IP address of a computer connected to the internet from a name.
>
> The A in A record stands for Address. Whenever you visit a web site, send an email, connect to Twitter or Facebook or do almost anything on the Internet, the address you enter is a series of words connected with dots.

Knowing this, we only need to configure our elastic IP in our domain. Take a look to the next image to see how this simple process is done.

```
sanguita@orion dig anuci.us

; <<>> DiG 9.10.4-P4-RedHat-9.10.4-2.P4.fc24 <<>> anuci.us
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 20529
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;anuci.us.                      IN    A

;; AUTHORITY SECTION:
us.                    900     IN    SOA   a.cctld.us. hostmaster.neustar.biz. 2017974937
900 900 604800 86400

;; Query time: 35 msec
;; SERVER: 212.142.144.66#53(212.142.144.66)
;; WHEN: Sun Dec 25 17:18:27 CET 2016
;; MSG SIZE  rcvd: 103
```

Automating deployments

For automating deployments, I decided to go with Jenkins but since I was using a dockerized deployment, WAR deployment was not so easy as running some commands from jenkins. Thus, using Jenkins was rejected and I created a little scripts that does the same thing, but however, it has to be manually executed.

The script is the next one, and it takes care of compiling, testing, building and deploying each new server update on the production server.

```
#!/bin/bash

#go to api

cd api
/opt/apache-maven-3.3.9/bin/mvn clean
/opt/apache-maven-3.3.9/bin/mvn install

#move file to tomcat webapps dir
mv ./api/target/*.war ./docker/Dockerfile/apps/

#go back
cd ..

#go to app
cd app

/opt/apache-maven-3.3.9/bin/mvn clean
/opt/apache-maven-3.3.9/bin/mvn install

#go back
cd ..

#move file to tomcat webapps dir
mv ./app/target/*.war ./docker/Dockerfile/tomcat/apps/

#run docker
cd docker

docker-compose up --build
```