Deployment of GoodGO Platform on AWS

O.O.3	**Deployment of GoodGO Platform on AWS**	Azione	**3.2**
Partner Responsabile	**GEOSOLUTIONS S.A.S.**		
Autori	Lorenzo Pini, Giovanni Allegri, Simone Giannecchini		
Nome del file	Deliverable_3.2.5.pdf	Stato	**definitivo**

Cronologia di approvazione del documento:

Data	Stato (Bozza/Revisione/Finale)	Autore/Revisore
12.12.2018	Bozza	Lorenzo Pini, Giovanni Allegri
20.12.2018	Revisione/finale	Simone Giannecchini

Scopo del documento

Objective of this document is to detail exactly the installation and deployment steps for the GoodGO platform on the AWS cloud.

In the upcoming sections, detailed instructions are provided for the setup which could be followed in order to deploy a separate instance.

Destinatari del documento

- ☐ OP Leaders
- ☒ Partners
- ☐ Associates
- ☒ Stakeholders
- ☐ Decision Makers
- ☐ Altri _____

Tipo di documento

- ☐ Private
- ☒ Non private
- ☐ Public

INDICE

Deployment of GoodGO Platform on AWS

1 INTRODUZIONE

Objective of this document is to detail exactly the installation and deployment steps for the GoodGO platform on the AWS cloud. In the upcoming sections, detailed instructions are provided for the setup which could be followed in order to deploy a separate instance.

Note for the italian readers
Il documento è stato redatto in inglese per facilitare la gestione e la manutenzione dei software da parte di team che includono personale non di lingua italiana.

2 OVERALL DEPLOYMENT DIAGRAM

In the Picture below (Figure 1) the current deployment of the Good Go platform on the AWS cloud platform is depicted.

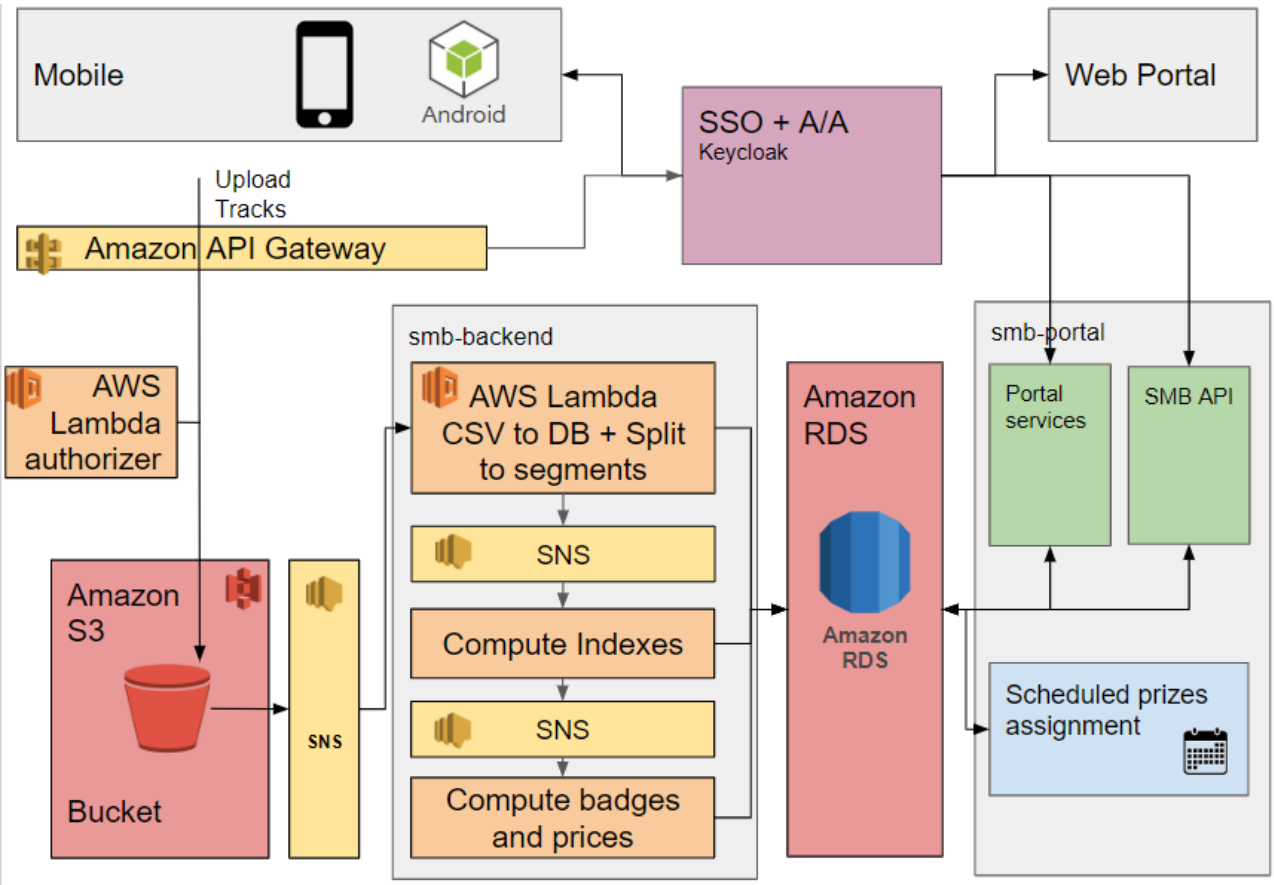

Figure 1. Current deployment on AWS

- **SSO:** the SSO service provides the Identity Management and the authentication services to the Web Portal and the mobile apps. The AuthN protocol is OpenID Connect.
- **Amazon S3**: this is the storage for the raw data uploaded by the mobile apps. The mobile app doesn't upload data directly to the S3 bucket, rather it goes through a lambda function (served by the Amazon API Gateway) that manages the placement of the raw files inside S3. An SNS signal triggers the data processing.
- **smb-backend**: this is the stack of lambda functions that are in charge of validating and processing users generated data, doing the indexes calculations, assigning badges and generating the ledger for the prizes based on users' performances. The event-based triggering and the communication between the lambdas is mediated by the AWS SNS service.

- **Amazon RDS**: this is the Amazon service providing the central DB (PostgreSQL) of the GoodGo and SMB infrastructure. It provides a HA and fault tolerant PostgreSQL DB, with automatic backups and failover systems, kept secured under the Amazon network security system.
- **smb-portal:** this is the container of the two main applications endpoints, the Portal services (which provides the public Web Portal) and the SMB (REST) API which provides the services to the mobile applications and back-channels to third party applications that complete the SMB infrastructure.
- **Web Portal**: this is the public portal where users can find information on GoodGo, manage their subscriptions and their bikes.

3 AWS SERVICES

3.2 API Gateway

The mobile app sends collected points to the /upload or /upload-dev endpoints

The API Gateway acts as a proxy, putting the files to a specific bucket, in a specific folder.
They are currently configured to push to:

- /upload/{folder} → smb-test-lamb/cognito/smb/{iamsub}/{object}
- /upload-dev/{item} → smb-user-uploads-dev/{iamsub}/{object}

NOTE: **{folder}** is not actually a folder, but the uploaded zip file, the name is an old usage of that API (the "dev" is correctly named **{item}**)
Authorization
The API endpoints allow access only if the requests have a valid Keycloak token.
In AWS terms, this validation is done by an authorizer.
The authorizer uses a lambda function to validate the token.

The token is passed in the "Authorization" header.
The authorizer name is: Keycloak-lambda-token
The lambda function used is: SaveMyBike-Keycloak-authorizer
The lambda function code is available at: https://github.com/geosolutions-it/keycloak-lambda-authorizer

In order to verify the token, the function must have the public key of the Keycloak instance that issued the token.
Follow the readme file to get the public key of the Keycloak instance and set it in the lambda function.
In short:

- Get the public key from the Keycloak server: https://your.server/auth/realms/your-realm/protocol/openid-connect/certs
  - Note: you have to use one of the JSON objects from the "keys" array
- *echo "JWT_SECRET=your-secret" > .env* to save the public key into a .env file
- Zip and upload to a new lambda function. (In this case is SaveMyBike-Keycloak-authorizer)

3.3 S3 Buckets

The buckets react to new files by posting to an SNS Topic, see below

Environment	S3 Bucket name
Dev	geosolutions-it-smb-test
Production (goodgo)	geosolutions-it-smb-production

3.4 SNS Topics

Each S3 bucket is configured to publish a notification into an SNS topic whenever new data is created

Environment	S3 Bucket that publishes notification	SNS Topic name
Dev	geosolutions-it-smb-test	smb-backend-dev
Production (goodgo)	geosolutions-it-smb-production	smb-backend-production

3.5 Lambda Functions

Each environment has a specific lambda alias. Each lambda alias can point to different versions of the same lambda. In each version of a lambda, the environment if frozen and cannot be changed

Environment	lambda alias	SNS topic consumed	lambda function	code
Dev	$LATEST	smb-backend-dev	`savemybike-smbbackend` - any snapshot of the git repository	https://github.com/geosolutions-it/smb-backend/tree/master/smbbackend
Production (goodgo)	production	smb-backend-production	`savemybike-smbbackend` - a tested and known to work commit sha of the repository	https://github.com/geosolutions-it/smb-backend/tree/master/smbbackend
Production			SaveMyBike-Keycloak-authorizer	https://github.com/geosolutions-it/keycloak-lambda-authorizer
Dev			SaveMyBike-Keycloak-authorizer-dev	https://github.com/geosolutions-it/keycloak-lambda-authorizer

4 KEYCLOACK

Keycloak is running in standalone mode, using version 4.1.0.Final, downloaded from:

https://downloads.jboss.org/keycloak/4.1.0.Final/keycloak-4.1.0.Final.tar.gz

Filesystem path	Description
/opt/keycloak-4.1.0.Final	Installation dir
/opt/keycloak-4.1.0.Final/standalone/configuration/standalone.xml	Main configuration file for Keycloak
/opt/keycloak-4.1.0.Final/bin/launch.sh	Execution script. This is run by systemd
/etc/systemd/system/keycloak.service	Systemd service file
/etc/keycloak/keycloak.conf	Systemd environment file

4.2 Database backend

Keycloak is configured to use the postgresql instance on RDS (DB and user creation steps outlined below, in RDS section).

Follow the steps outlined in the keycloak docs at

https://www.keycloak.org/docs/latest/server_installation/index.html#_database

For installing the postgresql JDBC driver:

- Installed JDBC driver from ubuntu's repository
- Added the relevant directory structure to keycloak's main directory
- Created the required driver file

```
> sudo apt install libpostgresql-jdbc-java
> cd /opt/keycloak-4.1.0.Final
> mkdir -p modules/system/layers/keycloak/org/postgresql/main
> cat << EOF > modules/system/layers/keycloak/org/postgresql/main/module.xml
> <?xml version="1.0" ?>
> <module xmlns="urn:jboss:module:1.3" name="org.postgresql">
>         <resources>
>         <resource-root path="/usr/share/java/postgresql-9.4.1212.jar"/>
>         </resources>
>
>         <dependencies>
>         <module name="javax.api"/>
>         <module name="javax.transaction.api"/>
>         </dependencies>
> </module>
> EOF
```

Then followed the keycloak guide for declaring and loading the postgresql driver in the main configuration file, by adding these:

```
<subsystem xmlns="urn:jboss:domain:datasources:4.0">
    <datasources>
      ...
      <drivers>
```

```
        <driver name="postgresql" module="org.postgresql">
            <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
        </driver>
        ...
      </drivers>
    </datasources>
  </subsystem>
```

```
 <subsystem xmlns="urn:jboss:domain:datasources:4.0">
    <datasources>
      ...
      <datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS" enabled="true" use-java-context="true">
        <connection-url>jdbc:postgresql://localhost/keycloak</connection-url>
        <driver>postgresql</driver>
        <pool>
          <max-pool-size>20</max-pool-size>
        </pool>
        <security>
          <user-name>keycloak-user</user-name>
          <password>xxxxxxxxxxxxx</password>
        </security>
      </datasource>
      ...
    </datasources>
  </subsystem>
```

4.3 Network setup

4.3.1 Proxy setup

This Keycloak instance is fronted by nginx which does SSL termination. As such, the SSL configuration has been done as described in the proxy section of the keycloak server docs:

https://www.keycloak.org/docs/latest/server_installation/index.html#_setting-up-a-load-balancer-or-proxy

Made the following modifications to configuration.xml:

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
    <http-listener name="default" socket-binding="http" redirect-socket="proxy-https"
      proxy-address-forwarding="true"/>
    ...
  </server>
  ...
</subsystem>
```

```
<socket-binding-group ...>
  ...
  <socket-binding name="proxy-https" port="443"/>
  ...
</socket-binding-group>
```

Nginx configuration is setup as a simple proxy, with all https requests to /auth/ being redirected to keycloak.

```
# ~/staging/nginx.conf
upstream staging_keycloak_server_ssl {
        server 127.0.0.1:8180;
}

server {
   ...

   location /auth/ {
        proxy_pass http://staging_keycloak_server_ssl;

        proxy_http_version 1.1;

        proxy_set_header Host               $host;
        proxy_set_header X-Real-IP          $remote_addr;
        proxy_set_header X-Forwarded-For    $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto  $scheme;
        }

}
```

4.3.2 *Socket port bindings*

Configured all keycloak network ports with an offset of 100 in order to avoid clashing with the instance used for dev. Changed the following on the configuration.xml file:

```
  <socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:100}">
      ...
   </socket-binding-group>
```

Note that this means that main port for this keycloak instance is 8180

4.3.3 *Outgoing HTTPS requests truststore*

Reusing the same truststore created for the dev instance (See instructions above). Copied the following into standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:1.1">
    <spi name="truststore">
            <provider name="file" enabled="true">
               <properties>
                  <property name="file" value="/home/geosolutions/keycloaktruststore.jks"/>
                  <property name="password" value="xxxxxx"/>
                  <property name="hostname-verification-policy" value="WILDCARD"/>
                  <property name="disabled" value="false"/>
               </properties>
            </provider>
         </spi>
</subsystem>
```

4.4 Login theme for SMB portal

Copied the contents of the keycloak_theme dir on the repo to the keycloak themes dir

```
> cp -r ~/staging/smb-portal/keycloak-theme/smb /opt/keycloak-4.1.0.Final/themes/
```

4.5 Running

Keycloak instance is managed by systemd

```
> sudo systemctl enable keycloak.service  # service restarts at boot
> sudo systemctl status|start|stop keycloak.service  # inspect, start or stop keycloak
> sudo journalctl [-f] -u keycloak.service  # check logs
```

5 RDS Database

5.2 RDS - Staging DB

RDS instance: savemybike-dev.cwv06fiycheq.us-west-2.rds.amazonaws.com
DB: savemybike-staging

Database was set up with the following:

```
RDS_HOST="savemybike-dev.cwv06fiycheq.us-west-2.rds.amazonaws.com"
RDS_USERNAME="main"
SMB_DB_NAME="savemybike-staging"
SMB_DB_USERNAME="savemybike-staging-user"
SMB_DB_PASSWORD="************************"

# password generated beforehand with:
# cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1

psql --host ${RDS_HOST} --username ${RDS_USERNAME} postgres << EOF
\set ON_ERROR_STOP true
DROP USER IF EXISTS "${SMB_DB_USERNAME}";
CREATE USER "${SMB_DB_USERNAME}" WITH PASSWORD '${SMB_DB_PASSWORD}';
GRANT "${SMB_DB_USERNAME}" TO ${RDS_USERNAME};
CREATE DATABASE "${SMB_DB_NAME}" WITH OWNER "${SMB_DB_USERNAME}";
\c ${SMB_DB_NAME}
CREATE EXTENSION postgis;
EOF

echo "Testing installation..."
psql --host ${RDS_HOST} --username ${SMB_DB_USERNAME} ${SMB_DB_NAME} << EOF
SELECT postgis_full_version();
EOF
```

5.3 RDS - goodgo DB

RDS instance: savemybike-dev.cwv06fiycheq.us-west-2.rds.amazonaws.com
DB: savemybike-production

Database was set up with the same script as for the staging DB, replacing the SMB_DB_NAME, SMB_DB_USERNAME and SMB_DB_PASSWORD with the appropriate values

The same steps as in dev DB are required for the setup.

6 Outbound SMTP account

The email service uses the SMTP service provided by the top domain (savemybike.eu) ISP.

SMTP: mail.savemybike.eu:465
SSL: yes
email: goodgo@savemybike.eu
password: **********

7 SMB Portal

7.2 Folders

App folder	/home/geosolutions/goodgo/smb-portal
uWsgi conf file	/etc/uwsgi/apps-enabled/smbportal-goodgo.ini -> /home/geosolutions/goodgo/uwsgi.ini
Nginx conf file	/etc/nginx/sites-enabled/goodgo.conf -> /home/geosolutions/goodgo/nginx.conf
Keycloak	Keycloak - Production
RDS	RDS - Dev

7.3 System dependencies

7.3.1 Extra ppas

> sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
> sudo add-apt-repository ppa:webupd8team/java

7.3.2 Apt packages

> sudo apt-get install \
> gettext \
> libgdal-dev \
> nginx \
> oracle-java8-installer \
> oracle-java8-set-default \
> postgresql \
> postgis \
> python3-pip \
> uwsgi \
> uwsgi-plugin-python3

7.4 Python

Installed virtualenvwrapper and created the smb-portal-prod virtualenv

> sudo -H pip3 install --upgrade pip vitualenvwrapper

Virtualenv

Python3 venv in /home/geosolutions/.virtualenvs/smb-portal-prod

Activate with
>workon smb-portal-prod

7.4.1 Python requirements

>pip install requirements/dev.txt

7.4.2 Env variables

These are all defined in the ~/.env file

DJANGO_SETTINGS_MODULE=base.settings.prod
DJANGO_ALLOWED_HOSTS=”127.0.0.1 localhost goodgo.savemybike.geo-solutions.it”
DJANGO_DEBUG=true
DJANGO_SECRET_KEY="**********"
DJANGO_DATABASE_URL=postgis://****:*****@savemybike-production.cwv06fiycheq.us-west-2.rds.amazonaws.com:5432/savemybike
DJANGO_PUBLIC_URL=https://goodgo.savemybike.geo-solutions.it
DJANGO_EMAIL_HOST="mail.savemybike.eu"
DJANGO_EMAIL_USE_SSL=true

DJANGO_EMAIL_PORT=465
DJANGO_EMAIL_HOST_USER="goodgo@savemybike.eu"
DJANGO_EMAIL_HOST_PASSWORD="*****"
KEYCLOAK_BASE_URL=https://goodgo.savemybike.geo-solutions.it
KEYCLOAK_REALM=save-my-bike
DJANGO_KEYCLOAK_CLIENT_ID=smb-portal
KEYCLOAK_ADMIN_USERNAME=”save-my-bike-admin”
KEYCLOAK_ADMIN_PASSWORD=”*****”
FCM_SERVER_KEY="*******"

A convenient way to load these when working remotely on the dev server is to run the following command:

> set -o allexport; source ~/.env; set +o allexport

It will export all variables into the current shell’s environment

7.5 uWsgi

```
[uwsgi]
env = DJANGO_SETTINGS_MODULE=base.settings.base
env = DJANGO_ALLOWED_HOSTS=127.0.0.1 localhost goodgo.savemybike.geo-solutions.it
env = DJANGO_DEBUG=true
env = DJANGO_SECRET_KEY="******"
env = DJANGO_DATABASE_URL=postgis://*****:*****@savemybikeproduction.cwv06fiycheq.us-west-2.rds.amazonaws.com:5432/savemybike
env = DJANGO_PUBLIC_URL=https://goodgo.savemybike.geo-solutions.it
env = DJANGO_EMAIL_HOST_USER=phony@mail.com
env = DJANGO_EMAIL_HOST_PASSWORD=1234
env = KEYCLOAK_BASE_URL=https://goodgo.savemybike.geo-solutions.it
env = KEYCLOAK_REALM=save-my-bike
env = DJANGO_KEYCLOAK_CLIENT_ID=smb-portal
env = KEYCLOAK_ADMIN_USERNAME=save-my-bike-admin
env = KEYCLOAK_ADMIN_PASSWORD=*****
env = FCM_SERVER_KEY=*****



socket = 127.0.0.1:3032
#socket = /var/run/uwsgi/smbportal/smbportal.socket
uid = geosolutions
gid = geosolutions
chdir = /home/geosolutions/goodgo/smb-portal/smbportal
wsgi-file = base/wsgi.py
virtualenv = /home/geosolutions/.virtualenvs/smb-portal-prod
processes = 2
enable-threads = true
threads = 2
buffer-size = 524288
autoload = false
touch-reload = base/wsgi.py
plugins = python3
master = true
max-requests = 50
```

7.6 Nginx

```
upstream smbportal-goodgo {
   server 127.0.0.1:3032;
}

# Expires map
map $sent_http_content_type $expires {
   default                  off;
   text/html                epoch;
   text/css                 max;
   application/javascript   max;
   ~image/                  max;
}

server {
      listen 80;
      server_name goodgo.savemybike.geo-solutions.it;
      charset utf-8;
      root /home/geosolutions/goodgo/smb-portal/static_root/;

      listen 443 ssl;

      ssl on;
      ssl_certificate /home/geosolutions/goodgo.savemybike.geo-solutions.it/goodgo.chained.crt;
      ssl_certificate_key /home/geosolutions/goodgo.savemybike.geo-solutions.it/goodgo.key;

      ##
      # Gzip Settings
      ##
      gzip on;
      gzip_vary on;
      gzip_proxied any;
      gzip_http_version 1.1;
      gzip_disable "MSIE [1-6]\.";
      gzip_buffers 16 8k;
      gzip_min_length 1100;
      gzip_comp_level 6;
      gzip_types video/mp4 text/plain text/css application/x-javascript text/xml application/xml application/xml+rss
text/javascript image/jpeg application/javascript;

      #gzip on;
      #gzip_min_length 800;
      #gzip_types text/plain application/json application/xml text/css text/javascript application/javascript;
      #gzip_comp_level 4;

      ##
      # Browser Cache Settings
      ##
      etag on;
      expires $expires;
      proxy_read_timeout 600s;

      # set client body size to 2M #
      #client_max_body_size 5000M;
      client_max_body_size 50M;

      location @uwsgi {
            etag off;
```

```
        uwsgi_pass  smbportal-goodgo;
        include     uwsgi_params;
        uwsgi_read_timeout 600s;
    }

    location /static/ {
       alias /home/geosolutions/goodgo/smb-portal/static_root/;
    }

    location /media/ {
       alias /home/geosolutions/goodgo/smb-portal/media/;
    }

    location /.well-known/ {

       alias /var/www/html/.well-known/;
    }


    location / {
        try_files $uri @uwsgi;
    }
}
```