To deploy the application into any environment of your choice, please follow the below steps.

Our environment:

- Our solution is hosted on Microsoft Azure cloud, and on Linux CentOS 7.2 operating system
- MySQL 5.7 on Linux CentOS 7.2
- Web and app servers on Linux CentOS 7.2
- CA approved SSL Certificates

Getting the environment ready

- Install Docker on both web and app servers
- Install MySQL 5.7 on Linux CentOS 7.2
- Login to Microsoft Azure private Registry hosted on Microsoft cloud from web server and app server. This is where the Docker images for NGINX web application are stored for the prototype. Use the command below:

```
docker login smfregistry01-on.azurecr.io --username=smfregistry01 --
password=44yI+v=qqfVXONIPE4ypuRMFhk6kL9f7
```

- Once you have logged into the Azure registry, pull the Docker images on both web and app servers. This will pull the images from Azure registry.

  Login to the webserver and use the command below to pull the image on the webserver

```
docker pull smfregistry01-on.azurecr.io/smf02/nginx
```

  Login to the appserver and use the command below to pull the images on the app server

```
docker pull smfregistry01-on.azurecr.io/smf02/dropwizard
```

Running the web (NGINX) container on the web server:

Before running the NGINX container, please perform the following steps

1) Open ports 443 and port 80 on the webserver
2) Place your certificates on /etc/ssl/certs/ directory on the server
3) Place your server key file on /etc/ssl/private/ directory on the server that is generated during CSR file generation
4) Create a .env file in the folder where you want to run the Docker container from which contains the app server information

   Here is the sample .env file, replace the appserver.com with your app server information

```
.env
REST_API_SERVER=https://appserver.com
REST_BASE_URL=appserver.com
REST_API_PORT=:443
```

5) Replace the default.conf file in the Docker container with the SSL certificate information and other information you may need, such as redirecting http to https. server_name will be your domain.com

Sample default.conf file

```
server {
    listen       80;
    server_name  example.com;
    return         301 https://example.com$request_uri;

    #charset koi8-r;
    #access_log  /var/log/nginx/log/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}
server
{
    listen       443;
    server_name  example.com;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
```

```
        }
        ssl    on;
        ssl_certificate    /etc/ssl/certs/mycalerts.crt;
        ssl_certificate_key    /etc/ssl/private/server.key;

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
```

6) Run the command below on web server. This will create and run a **web container** with the name "nginx". The command tells the Docker container to use ports 80 and 443, to mount the SSL certificates from /etc/ss/certs and /etc/ssl/private on the server into the Docker container. Replace anywhere that has http://localhost with REST_API_SERVER variable mentioned in the .env file. Replace anywhere where it says localhost with REST_BASE_URL variable mentioned in the .env file. Replace where it says port 8080 with REST_API_PORT variable in the .env file.

```
sudo docker run -dit --restart=always --name=nginx -p 80:80 -p 443:443 -v
/etc/ssl/certs:/etc/ssl/certs -v /etc/ssl/private:/etc/ssl/private --env-file .env
smfregistry01-on.azurecr.io/smf02/nginx /bin/bash -c $'sed -i
"s#http://localhost#$REST_API_SERVER#g" /usr/share/nginx/html/scripts/*.js && sed -
i "s#localhost#$REST_BASE_URL#g" /usr/share/nginx/html/scripts/*.js && sed -i
"s#:8080#$REST_API_PORT#g" /usr/share/nginx/html/scripts/*.js && nginx -g "daemon
off;"'
```

7) Copy the default.conf into the Docker container using the below command and restart the container.

```
sudo docker cp default.conf nginx:/etc/nginx/conf.d/default.conf
```

8) Restart NGINX container

```
sudo docker restart nginx
```

Running DropWizard Docker container on app server

1) Open 443 on the app server

2) Create a .env file in the folder where you want to run the Docker container from, This contains the database information and keystore_file information. Keystore can be generated as below.

   Sample .env file

```
 MYSQL_ROOT_USERNAME=
 MYSQL_ROOT_PASSWORD=
MYSQL_HOSTNAME=
MYSQL_PORT=
MYSQL_DB_NAME=
KEYSTORE_FILE= cgi-poc-dw.keystore(please use this name, don't change this)
KEYSTORE_PASSWORD=
```

3) Generate Keystore from SSL certs

- Concatenate the intermediate and ssl server certificate into one
  `cat ssl_certificate.cer IntermediateCA.cer >> drop.app.crt`

- This step is to convert them into a single PKCS12 file using the command
  `openssl.exe pkcs12 -export -in drop.app.crt -inkey server.key > dropwizard.p12`

- Record the password

- Import the PKCS12 file into a keystore using the command

  ```
  keytool -importkeystore -srckeystore dropwizard.p12 -destkeystore dropwizard.jks -
  srcstoretype pkcs12
  ```

- Rename the dropwizard.jks to cgi-poc-dw.keystore, since this name was used while building the DropWizard container.

  ```
  mv the dropwizard.jks to cgi-poc-dw.keystore
  ```

You now have a keystore named host.jks containing the certificate/key you need.

- Copy the cgi-poc-dw.keystore file to the same location as the .env file

4) Change the config.yml file as needed and place it in the same location as .env file and cgi-poc-dw.keystore. You have to change the cors section to allow your web server domain to access the app server ,SMTP server section, and Twilio section for sending messages.

   Sample config.yml file

```
jwtSignatureSecret:
o5KnhuFU8q8bo6qPxG2Z75q3yLlNiVCSOUd0vqbv6X8E8QLUSbP6Uxn7hn2678NZ3sQ1ZKy9ih7jAHXu

jwtExpiryInMinutes: 1440

# Database settings.
database:


    # the name of your JDBC driver
    driverClass: com.mysql.jdbc.Driver

    # the username
    user: ${MYSQL_ROOT_USERNAME}

    # the password
    password: ${MYSQL_ROOT_PASSWORD}

    # the JDBC URL
    url:
jdbc:mysql://${MYSQL_HOSTNAME}:${MYSQL_PORT}/${MYSQL_DB_NAME}?createDatabaseIfNotEx
ist=true

    # The initial size of the connection pool.
    initialSize: 2
```

```
    # The minimum size of the connection pool.
    minSize : 2

    # The maximum size of the connection pool.
    maxSize : 50

# Enable HTTPS
server:
    applicationConnectors:
        - type: http
          port: 8080
        - type: https
          port: 8443
          keyStorePath: ${KEYSTORE_FILE}
          keyStorePassword: ${KEYSTORE_PASSWORD}
          validateCerts: false
          validatePeers: false
          supportedProtocols: [TLSv1, TLSv1.1, TLSv1.2]
          excludedCipherSuites: [SSL_RSA_WITH_DES_CBC_SHA,
SSL_DHE_RSA_WITH_DES_CBC_SHA, SSL_DHE_DSS_WITH_DES_CBC_SHA,
SSL_RSA_EXPORT_WITH_RC4_40_MD5, SSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA, SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA]

#Jersey client settings
jerseyClient:
    #The maximum time to wait for a connection to be returned from the connection
pool.
    connectionRequestTimeout: 1000ms
    #The maximum time to wait for a connection to open.
    connectionTimeout: 1000ms
    #The maximum idle time for a connection, once established.
    timeout: 1000ms
    #The size of the work queue of the pool used for asynchronous requests.
    #Additional threads will be spawn only if the queue is reached its maximum
size.
    workQueueSize: 16
# CORS settings
#   A list of comma-separated domains that the service will allow cross-origin
access from.  If the Origin matches one
#     of the domains in the list the application will generate a matching Access-
Control-Allow-Origin header in the response.
#     If the Origin header does not match, the application will not send the A-C-A-
O header back to the client.
#   Uses Jetty's CrossOriginFilter, so partial regex is supported:
#     i.e. https?://*.cgi.[a-z]{3}   -> supports http/https with any subdomain and
any 3 character top-level domain for the host "cgi"
#   c.f. https://www.eclipse.org/jetty/documentation/9.3.x/cross-origin-filter.html
# IMPORTANT NOTE: if there is no * character in the expression, the filter
processes it as a raw string, NOT as a regex
cors:
```

```
  allowedDomains: https?://*.cgi.[a-z]{3},https?://*.cgi.[a-z]{3}:[0-
9]{4},http://localhost:9000,http://localhost:8000,https?://(192.168).*.*:[0-
9]{4},https?://(10.1).*.*:[0-9]{4}
  allowedHeaders:
   - Accept
   - Authorization
   - Content-Length
   - Content-Type
   - Content-Disposition
   - Cookie
   - Origin
   - Referer
   - x-amz-date
   - x-api-key
   - x-requested-by
   - x-requested-with

mapsApi:
  apiURL: http://maps.googleapis.com/maps/api/geocode/json
  apiKey: <your_key>

# Notification mail settings.
mail:

   #The host running the SMTP server to use.
  hostname:

   #The port at which the SMTP server listens on.
  port: 587

   #The address used as From for outgoing notification mails sent by the server.
  systemEmail:

   #The username used to access the mail server.
  username:

   #The password used to access the mail server.
  password:

   #Connect using SSL.
  ssl: true

   #Connect using TLS.
  tls: true

# Logging settings
logging:
  level: INFO
  loggers:
    com.cgi: DEBUG

# Swagger settings
swagger:
  resourcePackage: com.cgi.poc.dw.rest.resource
#Jobs settings
scheduler:
  #number of thread to use to run the jobs
```

```
  thread: 5
  #Define a job for an event API
  jobs:
    # the URL of the API event
     - eventType: Fire
       eventURL:
https://wildfire.cr.usgs.gov/arcgis/rest/services/geomac_dyn/MapServer/0/query?f=js
on&where=1%3D1&outFields=*&outSR=4326
       #the delay to start the job after the server start
       delay: 1
       #the delay between to running job's
       period: 900
       #the time unit to define the delay and the period
       timeUnit: SECONDS
      # the URL of the API event
     - eventType: Weather
       eventURL:
https://idpgis.ncep.noaa.gov/arcgis/rest/services/NWS_Forecasts_Guidance_Warnings/w
atch_warn_adv/MapServer/0/query?f=json&where=1%3D1&outFields=*&outSR=4326
       #the delay to start the job after the server start
       delay: 1
       #the delay between to running job's
       period: 900
       #the time unit to define the delay and the period
       timeUnit: SECONDS
     - eventType: Flood
       eventURL:
https://idpgis.ncep.noaa.gov/arcgis/rest/services/NWS_Observations/ahps_riv_gauges/
MapServer/0/query?f=json&where=(status%20%3D%20%27minor%27%20OR%20status%20%3D%20%2
7major%27%20OR%20status%20%3D%20%27moderate%27)%20AND%20(1%3D1)&spatialRel=esriSpat
ialRelIntersects&outFields=*&outSR=4326
       #the delay to start the job after the server start
       delay: 1
       #the delay between to running job's
       period: 21600
       #the time unit to define the delay and the period
       timeUnit: SECONDS
twilio:
  accountSID:
  authToken:
  phoneNumber:
```

5) Once you have all the steps above complete, use the commands to run the Docker container.

```
sudo docker run -dit  --restart=always --name=dropwizard -p 443:8443 --env-file
.env smfregistry01-on.azurecr.io/smf02/dropwizard ./dw serve
sudo docker cp cgi-poc-dw.keystore dropwizard:/usr/src/app/cgi-poc-dw.keystore
sudo docker cp config.yml dropwizard:/usr/src/app/config.yml
sudo docker restart dropwizard
```

The deployments to web and app servers are complete. Now you can test by going to the URL of the web server, for example, www.mycalerts.com.