
USER GUIDE

SMART GARBAGE MONITORING SYSTEM

Elaborated by:

RIHAB JERBI
rihab.jerbi@supcom.tn

BEDIS MANSAR
bedis.mansar@supcom.tn

Supervisor:

DR. MOHAMED BÉCHAA
KAÂNICHE
medbecha.kaaniche@supcom.tn

Academic year: 2022 - 2023

Contents

1	Introduction	1
2	Domain name	1
3	TLS letsencrypt certificate	1
4	MQTT Broker	2
5	Wildfly TLS setup	3
6	MongoDB	6
7	IOT devices	7
8	Deployment	8

List of Figures

1	Linking the domain with namecheap	1
2	DNS TXT records	2
3	Generating JWT keystore	6
4	Microprofile configuration	6
5	Electronic circuit	7
6	Red-node architecture	7

1 Introduction

This user guide explains how to reproduce this project. All installation steps, technologies used, hardware components and the deployment procedure are explained in great detail.

2 Domain name

A domain name is required for this project. To achieve that. These are the steps to follow to acquire a domain name:

Github Student Pack: All students can acquire with their school's mail a Github student pack which offers various services for free.

Namecheap: Namecheap is a domain name registrars that offers domain names at affordable prices. Github Student Pack grants free Namecheap services for a year. To acquire these services, access namecheap from github, register an account and follow the guideline to acquire a domain name.

3 TLS letsencrypt certificate

Letsencrypt is a free certificate authority providing X.509 certificates for Transport Layer security. Security is a crucial part of our project. Therefore, it is essential to generate a TLS certificate for the acquired domain. The first prerequisite is a Virtual Machine. In our case, we have benefitted from the free 100\$ voucher given by Microsoft Azure for Sup'Com students. The second prerequisite is linking the ip address of the Virtual machine to namecheap as illustrated by figure 1:

<input type="checkbox"/>	Type	Host	Value	TTL	
<input type="checkbox"/>	A Record	@	20.23.253.114	Automatic	
<input type="checkbox"/>	A Record	api	20.23.253.114	Automatic	
<input type="checkbox"/>	A Record	iam	20.23.253.114	Automatic	
<input type="checkbox"/>	A Record	mqtt	20.23.253.114	Automatic	
<input type="checkbox"/>	CAA Record	@	0 issue "letsencrypt.org"	Automatic	

Figure 1: Linking the domain with namecheap

The host "@" represents the main domain. "api", "iam" and "mqtt" represents subdomains. If you have multiple Virtual machines, you give each virtual machine's ip address to a subdomain of your choice. Now, access the virtual machine that is given the main domain and follow these steps:

1. **Install Certbot:** Certbot is a tool that facilitates fetching and deploying certificates over web servers. To install it, issue the following commands:

```
$ sudo snap install --classic certbot
```

```
$ sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

2. **Generate certificate:** With Certbot, generate a certificate for the domain and the corresponding subdomains:

```
$ sudo certbot certonly --manual --agree-tos --no-eff-email --staple-  
  ↳ ocsp --key-type=ecdsa --elliptic-curve=secp256r1 --preferred-  
  ↳ challenges dns --debug-challenges -d \*.smartgarbagecot.me -d  
  ↳ smartgarbagecot.me
```

When issuing this command, you will be given a value and prompted to deploy a DNS TXT record under the name `_acme-challenge.example.me` on namecheap as demonstrated by figure 2:


<input type="checkbox"/>	TXT Record	_acme-challenge	vAq-hRyjTIAE_Q6zQPGaC4jEIQ6TLTxKH43bzEMGi-Y	Automatic	
<input type="checkbox"/>	TXT Record	_acme-challenge	CiH78FVOutKjd6pPPj31AdKrgNatMI4XX5BH0y_ukqw	Automatic	

Figure 2: DNS TXT records

Test that the the records were deployed properly with DNS lookup. After this command, you now have a valid certificate with the `privkey.pem` and `cert.pem` files stored in `/etc/letsencrypt/live/smartgarbagecot.me/`

4 MQTT Broker

For our solution, sensors will communicate and send data to the middleware via the MQTT Broker. These are the steps to install and configure the open source MQTT Broker Mosquitto:

1. **Install Mosquitto:** To install Mosquitto, type the following commands:

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

```
$ sudo apt install wget apt-transport-https gnupg2 software-properties-  
  ↳ common
```

```
$ sudo add-apt-repository ppa:mosquitto-dev/mosquitto-ppa
```

```
$ sudo apt install mosquitto mosquitto-clients
```

You can verify that Mosquitto has been installed by typing `"mosquitto -version"`

2. **Create MQTT username and password:** To block the broker from unintended connections, create a user and password for this broker with the following command:

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd username
```

3. **Generate Dellphi-Helman parameters:** For more security measures, generate `Dh-param.pem` with the command below:

```
$ sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

4. **Configure MQTT ports and security:** To choose the ports for websocket connection and to use theletsencrypt certificates generated earlier, modify the mosquitto configuration file in `/etc/mosquitto/conf.d/default.conf`:

```
listener 1883
password_file /etc/mosquitto/passwd
listener 8883
certfile /etc/mosquitto/certs/fullchain.pem
cafile /etc/ssl/certs/ISRG_Root_X1.pem
keyfile /etc/mosquitto/certs/privkey.pem
dhparamfile etc/ssl/certs/dhparam.pem

listener 8083
protocol websockets
certfile /etc/mosquitto/certs/fullchain.pem
cafile /etc/ssl/certs/ISRG_Root_X1.pem
keyfile /etc/mosquitto/certs/privkey.pem
dhparamfile /etc/ssl/certs/dhparam.pem
```

5. **Restart the broker:** To save and enable these modifications, restart the broker:

```
$ sudo systemctl restart mosquitto
```

Verify that there were no issues in the configuration with the command:

```
$ sudo systemctl status mosquitto
```

With this, we can access the broker over SSL connection to the broker and in particular over websockets and SSL.

5 Wildfly TLS setup

Wildfly is the application server used for this application. It needs to be installed as a service and be configured properly for HSTS and TLS:

1. **Install JDK 17:** Wildfly will not run without Java installed on the machine. Type the command:

```
$ apt install openjdk-17-jdk openjdk-17-jre
```

verify that java has been installed by typing "java -version"

2. **Install Wildfly-preview 26.1.2:** Download wildfly from the official website and extract the file. Then, move the folder to `/opt/wildfly`:

```
$ sudo mv wildfly-${WILDFLY_RELEASE} /opt/wildfly
```

3. **Configure Systemd for Wildfly:** To run the Wildfly service, create a system user and group

```
$ sudo groupadd --system wildfly
$ sudo useradd -s /sbin/nologin --system -d /opt/wildfly -g wildfly
    ↪ wildfly
```

Create a configuration directory for Wildfly:

```
$ sudo mkdir /etc/wildfly
```

Copy wildfly configuration files and start scripts to the newly created directory:

```
$ sudo cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.conf /etc/
    ↪ wildfly/
$ sudo cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.service /etc/
    ↪ systemd/system/
$ sudo cp /opt/wildfly/docs/contrib/scripts/systemd/launch.sh /opt/
    ↪ wildfly/bin/
$ sudo chmod +x /opt/wildfly/bin/launch.sh
```

Set /opt/wildfly directory permissions:

```
$ sudo chown -R wildfly:wildfly /opt/wildfly
```

After all these configurations, reload systemd service:

```
$ sudo systemctl daemon-reload
```

Start the wildfly service:

```
$ sudo systemctl start wildfly
$ sudo systemctl enable wildfly
```

You can verify that the wildfly service is running properly by running:

```
$ sudo systemctl service wildfly
```

Add wildfly users:

```
$ sudo /opt/wildfly/bin/add-user.sh
```

When prompted on the type of user desired, select Management user and provide the username and password.

4. **Wildfly TLS configuration:** With wildfly service running properly, we can now configure it for TLS and HSTS. We will be using the same certificates generated for Mosquitto. The first step is to convert the certificates from .pem format to .jks format. We first convert them to .pfx format:

```
$ sudo openssl pkcs12 -export -out cert.pfx -inkey /etc/letsencrypt/live/
    ↪ smartgarbagecot.me/privkey.pem -in /etc/letsencrypt/live/
    ↪ smartgarbagecot.me/cert.pem -certfile /etc/letsencrypt/live/
    ↪ smartgarbagecot.me/chain.pem
```

In this command, you will be asked to type in a password for the certificate. Then, we convert the pfx format to jks format:

```
$ keytool -importkeystore -srckeystore cert.pfx -srcstoretype PKCS12 -  
    ↪ srcstorepass password -storepass password -destkeystore cert.jks -  
    ↪ deststorepass password
```

Replace "password" by the password you typed in the previous command. After this command, copy "cert.jks" and place it in jboss configuration directory:

```
$ sudo cp cert.jks /opt/wildfly/standalone/configuration
```

Go to wildfly/bin and run jboss-cli.sh. Type in connect and type in the management user credentials created in step 3. Once connected, we can configure TLS for wildfly with these commands:

```
/subsystem=elytron/key-store=bedjer:add(path=cert.jks,relative-to=jboss.  
    ↪ server.config.dir,credential-reference={clear-text="password"},type  
    ↪ =JKS)  
  
/subsystem=elytron/key-manager=bedjerksm:add(key-store=bedjer,credential-  
    ↪ reference={clear-text="password"})  
  
/subsystem=elytron/server-ssl-context=bedjersslcontext:add(key-manager=  
    ↪ bedjerksm,protocols=["TLSv1.3"],cipher-suite-names="  
    ↪ TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:  
    ↪ TLS_AES_128_GCM_SHA256")  
  
/subsystem=undertow/server=default-server/https-listener=https:write-  
    ↪ attribute(name=ssl-context,value=bedjersslcontext)  
  
/core-service=management/management-interface=http-interface:write-  
    ↪ attribute(name=ssl-context,value=bedjersslcontext)  
  
/core-service=management/management-interface=http-interface:write-  
    ↪ attribute(name=secure-socket-binding,value=management-https)
```

In these commands, we give the path of our cert.jks, configure the keystore and key-manager. We add the protocol "TLS1.3" with a strong cryptographic suite. The final command prompts the user to select to add the certificate with choices of "yes" or "no" or "permanent". Choose permanent. Once finished, we now have TLS1.3 configured for wildfly on port 8443. The final step is to configure the Https Strict transport security for maximum security as https forces only secure connections:

```
/subsystem=undertow/configuration=filter/response-header=hsts:add(header-  
    ↪ name=Strict-Transport-Security,header-value="max-age=63072000;␣  
    ↪ includeSubDomains;␣preload")  
  
/subsystem=undertow/configuration=filter/rewrite=http-to-https:add(target  
    ↪ ="https://%v:8443%U",redirect=true)  
  
/subsystem=undertow/server=default-server/host=default-host/filter-ref=  
    ↪ hsts:add(predicate="equals(%p,8443)")
```



```
/subsystem=undertow/server=default-server/host=default-host/filter-ref=  
↪ http-to-https:add(predicate="equals(%p,8080)")
```

Now, HSTS and TLS are properly configured on wildfly. Since our code utilizes JSON web tokens, we require a JWT keystore. To generate that keystore, run these commands:

```
$ openssl genrsa -out private.pem 4096
```

Convert the keystore to jks format with the following commands:

```
cot@cot:~/uploads$ openssl req -key private.pem -new -x509 -days 3650 -subj "/C=FI/ST=Helsinki/O=Rule of Tech/OU=Information unit/CN=ruleoftech.com" -out cert.pem  
cot@cot:~/uploads$ openssl pkcs12 -export -inkey private.pem -in cert.pem -out keys.pfx -name "my alias"  
Enter Export Password:  
Verifying - Enter Export Password:  
cot@cot:~/uploads$ keytool -importkeystore -srckeystore keys.pfx -srcstoretype PKCS12 -srcstorepass dabousa123# -storepass dabousa123# -destkeystore jwt.jks  
deststorepass dabousa123# -alias "jwt"  
Importing keystore keys.pfx to jwt.jks...  
keytool error: java.lang.Exception: Alias <jwt> does not exist  
cot@cot:~/uploads$ keytool -importkeystore -srckeystore keys.pfx -srcstoretype PKCS12 -srcstorepass dabousa123# -storepass dabousa123# -destkeystore jwt.jks
```

Figure 3: Generating JWT keystore

Finally, place the jwt.jks file in jboss configuration directory. The final configuration for wildfly is setting it up to read vital information such as mosquito password from the server application itself. To acquire these informations, in our code we have utilized microprofile config API. Microprofile Config API reads the parameters from the server application first, then from configuration parameters configured in system variables and finally from configuration parameters "microprofile-config.properties" in META-INF directory. It is best practise to place these variables either in the server application or in system variables. To place them in the server application, modify the standalone.xml file in /opt/wildfly/standalone/configuration, specifically "subsystem xmlns="urn:wildfly:microprofile-config-smallrye:2.0" as shown by the figure below:

```
<subsystem xmlns="urn:wildfly:microprofile-config-smallrye:2.0">  
  <config-source name="props">  
    <property name="mqtt.broker.username" value="cot"/>  
    <property name="mqtt.broker.password" value="Granturismo123@"/>  
    <property name="mqtt.broker.broker" value="wss://mqtt.smartgarbagecot.me:8083"/>  
    <property name="mqtt.broker.clientId" value="mqtt_listener"/>  
    <property name="document" value="document"/>  
    <property name="document.database" value="document"/>  
    <property name="document.settings.jakarta.nosql.host" value="localhost:27017"/>  
    <property name="document.provider" value="org.eclipse.jnosql.diana.mongodb.document.MongoDBDocumentConfiguration"/>  
    <property name="argon2.saltLength" value="32"/>  
    <property name="argon2.hashLength" value="128"/>  
    <property name="argon2.iterations" value="23"/>  
    <property name="argon2.memory" value="97579"/>  
    <property name="argon2.threadNumber" value="2"/>  
    <property name="key.pair.lifetime.duration" value="10800"/>  
    <property name="key.pair.cache.size" value="3"/>  
    <property name="jwtTokenValidity" value="2020"/>  
    <property name="jwtIssuer" value="smartgarbagecot.me"/>  
    <property name="jwtAudience" value="smartgarbagecot.me"/>  
    <property name="jwtClaimRoles" value="groups"/>  
    <property name="jwtSecret" value="dabousa123#"/>  
    <property name="jwtAlias" value="jwt"/>  
  </config-source>  
</subsystem>
```

Figure 4: Microprofile configuration

6 MongoDB

MongoDB must be installed on the virtual machine as it will serve as the database for the whole project. These are the steps to install mongodb:

```
$ sudo apt install -y mongodb
```

```
$ echo "deb[arch=amd64,arm64]https://repo.mongodb.org/apt/ubuntu/focal/mongodb-org/5.0multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
$ sudo apt update
$ sudo apt install -y mongodb-org
```

Verify the installation:

```
$ mongod --version
```

Start mongodb as a service:

```
$ sudo systemctl start mongod
```

7 IOT devices

The figure below showcases the electronic circuit for this solution featuring one sensor:

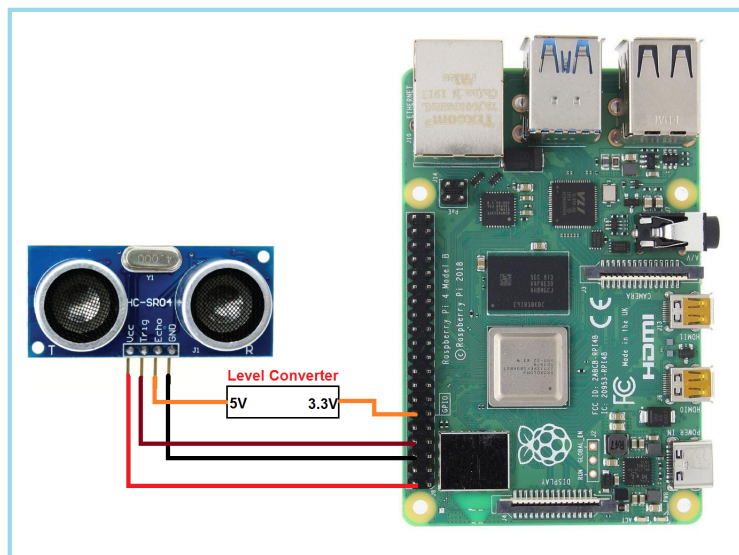


Figure 5: Electronic circuit

The sensor used is ultrasonic sensor HC-SR04 which can measure the distances. It will be used to measure the distance between the garbage lid and the waste inside it. To send that data to the Mqtt broker, node-red is used. Figure 6 displays the architecture of the node-red to send data to the mosquitto broker:

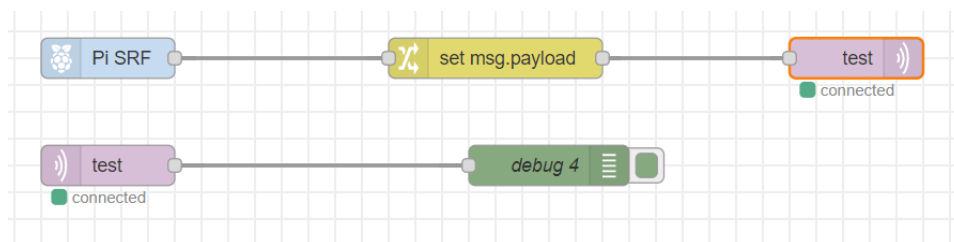


Figure 6: Red-node architecture

We have generated and stored a flows.json file containing this architecture and it can be modified accordingly if you setup the sensor in other pins.

8 Deployment

To run and deploy the achieved work, follow these steps:

1. Clone the Github repository.
2. Load the flows.json file located in IOT folder into your node-red in raspberry.
3. Create a "microprofile-config.properties" file under META-INF directory and store in it the values indicated in figure 4.
4. Create the JWT keystore.
5. Package the middleware folder into a single .war file with IntelliJ and put in in /wildfly/standalone/deployments.
6. Copy the flutter apk in the "apk" folder and put it in your phone to test.