Vanilla CAs How To

Shirley Crompton, UKRI-STFC

Purpose

The vanilla CAs web application provides basic certification services to support the Year 2 demonstration of the mF2C project. It should be noted that the application is not intended for production use.

The application is written in JAVA (v. 1.8.0_162-b12) using the Jersey library (v. 2.27) and is deployed as an REST application on a Tomcat 8 (v 8.5.35) server running in a Docker (v. 18.03.0-ce) container.

The first part of the document provides information on how to build and deploy the application on the Engineering VM (running CENTOS 7.3). The second part gives examples on how to use the application. Please note that I use a Windows machine, please adjust the commands and path elements as appropriate to suit your environment.

Installation Notes

The user is assumed to have the appropriate privileges to access and deploy artifacts on the Engineering machine (213.205.14.13). The installation steps are as follows:

1 Create the docker build folder and assemble the artefacts:

1.1 Create the folder 'certauths', a subfolder each for 'tomcat' and 'credentials'. (The 'certauths' folder is currently located in /usr/local/libexec/catShirley/)

1.2 Copy the credential files (*.key, *.pem, tomcat*) from ownclouds (mF2C\Working Folders\WP5 PoC integration\CA\CA credentials\) to the local credentials folder. If you need a new tomcat server certificate, see the section on 'Generating a Tomcat Server Certificate' lower down.

1.3 Copy the tomcat server.xml and catalina.sh files from ownclouds (mF2C\Working Folders\WP5 PoC integration\CA\tomcat\) to the local tomcat folder

1.4 Clone the certauth application from the mf2c git (https://github.com/mF2C/certauth.git):

[certauths] clone https://github.com/mF2C/certauth.git ./certauth

1.5 Build the Docker image, provide the correct tag version (e.g. mf2c/certauths:v1) as required:

[certauths] docker build -f ./certauth/Dockerfile -–rm –t mf2c/certauths:v1.02 .

1.6 Run the image and create a container with the specific containerName, we want to map the container's port 8443 to the host's port 54443

~~[certauths] docker run –t –p 8080:8080 –-name <containerName> mf2c/certauths:v1~~

[certauths] docker run –t –p 54443:8443 –-name <containerName> mf2c/certauths:v1.02

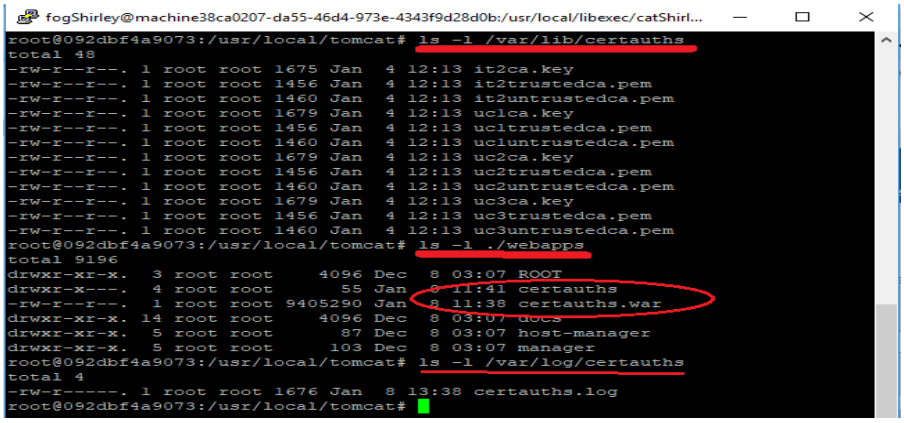
The Tomcat console output should indicate that the certauths.war has been deployed.

1.7 Cntr c to exit the running container. You could check the service endpoints using the methods listed in the Usage Notes section.

1.8 Verify the installation in the container via an interactive session. Start the session:

[certauths] docker exec –it <container name> /bin/bash

1.8.1 Check that the credential files have been copied etc. Note that the command can only be used against a running container and that the log file will only be created after a call has been made to the endpoint.



You could also use curl commands against the localhost name to test the service (see examples in the Usage notes section).

1.8.2 Exit the interactive session by typing 'exit'

1.9 Next, we want to configure the firewall, we need to obtain the IP address of the container. First, get the container ID by listing the currently running containers, use the container name to locate the correct container:

[certauths] docker ps

1.10 Check the local IP for the container

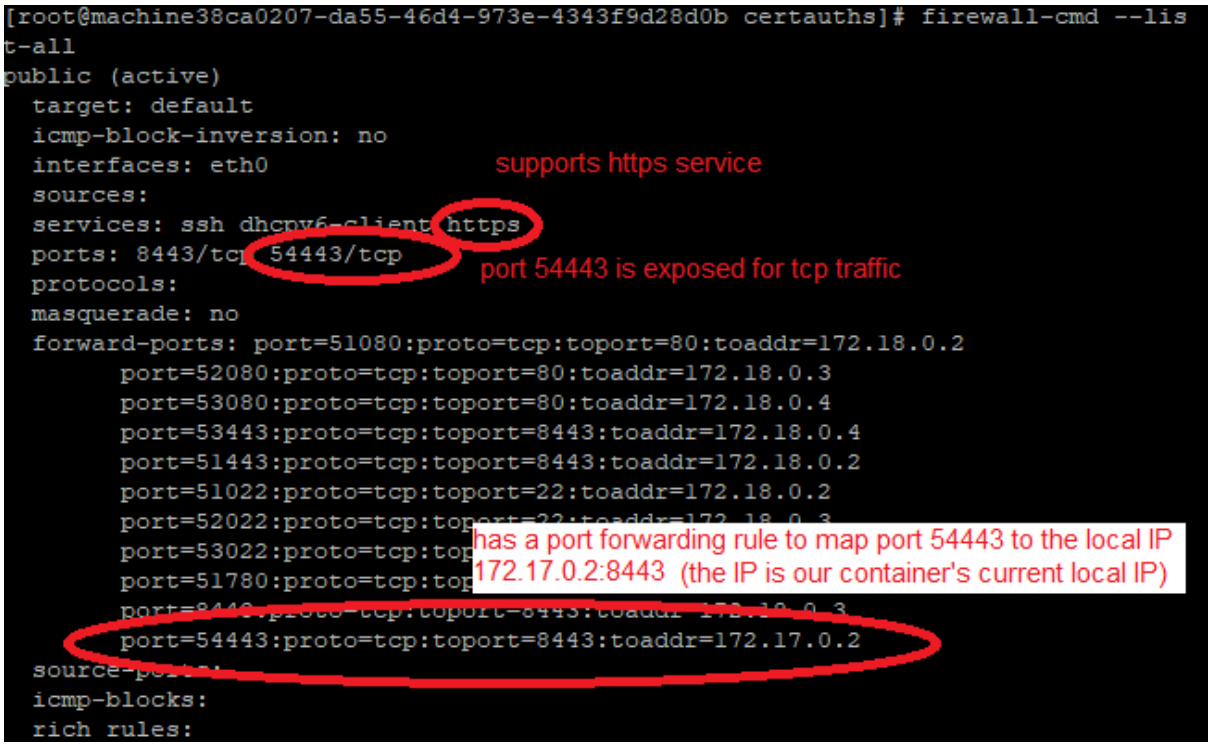
[certauths] docker inspect -f ip='{{.NetworkSettings.IPAddress}}' <containerID>

1.11 You may also like to verify the port mapping:

[certauths] docker inspect –f ports='{{.NetworkSettings.Ports}}' <containerID>

1.12 Check the firewall using firewall-cmd to see if it has the appropriate configuration:

[certauths] firewall-cmd –list-all



Check that it has the above settings. If not, update the configuration to match. See https://www.thegeekdiary.com/5-useful-examples-of-firewall-cmd-command/

1.13 Upload the image to Docker hub:

1.13.1 Login to Docker hub and follow the on-screen instructions:

[certauths] docker login

1.13.2 Stop the running container

[certauths] docker stop <container id/container name>

1.13.3 Commit

[certauths] docker commit –m “my commit message” –a “Author Name” <containername> <imagename>

1.13.4 Push to Docker hub

[certauths] docker push <image-name/version>

1.13.5 Restart the container

[certauths] docker start <container id/container name>

Generating a Tomcat Server Certificate

Use the IT2trustedca service to issue a certificate for your server, provide the IP of your server as input (see the Usage Notes on how to do this). Then we need to use OpenSSL to generate a self-contained certificate in PKCS12 format:

<OpenSSL> pkcs12 -export -in <x509.pem> -inkey <RSAprivate.key> -out tomcat.p12 -name tomcat -CAfile <path\it2TrustedCA.pem> -caname root –chain

Then we use JAVA keytool to convert the tomcat.p12 file into a JAVA keystore. Enter a passphase when prompted, using the same one throughout (this passphrase is used in the server.xml connector configuration):

"%JAVA_HOME%"\bin\keytool -importkeystore -v -srckeystore <path\tomcat.p12> -srcstoretype PKCS12 -destkeystore <\path\tomcat.keystore> -deststoretype pkcs12

Next, check the keystore, enter the passphrase you used before as store pass:

"%JAVA_HOME%"\bin\keytool -list -keystore <path\tomcat.keystore>

Usage Notes

The webapp context root of the Tomcat server on the Engineering machine is:

~~http://213.205.14.13:8080/ or http://it1demo.mf2c-project.eu:8080/~~
https://213.205.14.13:54443/ or https://it1demo.mf2c-project.eu:54443/

The server has been updated to accept https connection only. Please use https protocol and the updated context root for the usage examples below.

The vanilla CA web application provides four sets of 2 endpoints with each set supporting a specific IT-2 demo scenarios:

1 ‘Hello World’ IT-2 demo:

1.1 https://213.205.14.13:54443/certauths/rest/it2trustedca

1.2 https://213.205.14.13:54443/certauths/rest/it2untrustedca

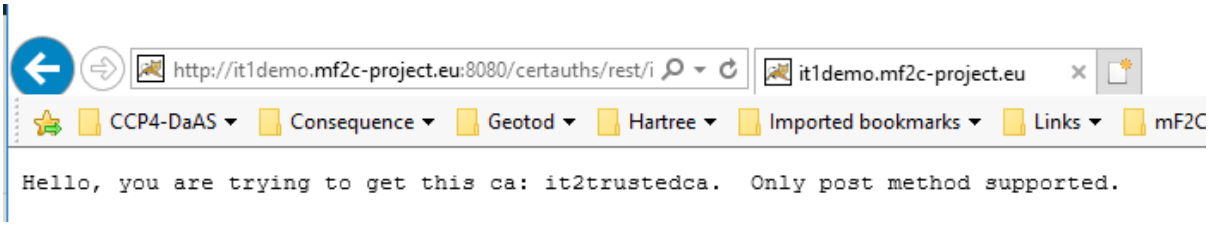
2 UC1 Smart City demo:

2.1 https://213.205.14.13:54443/certauths/rest/uc1trustedca

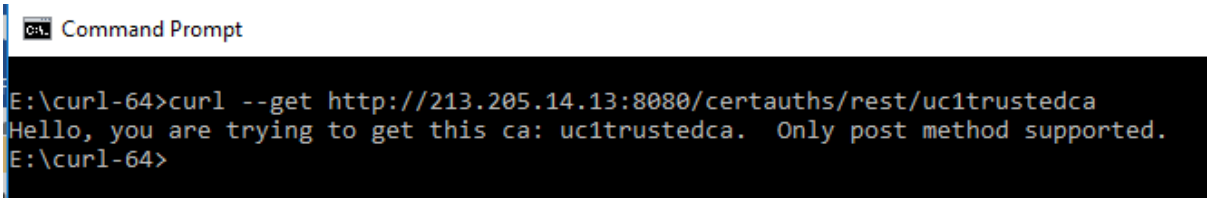
2.2 https://213.205.14.13:54443/certauths/rest/uc1untrustedca

3 UC2 Smart Boat demo:
3.1 https://213.205.14.13:54443/certauths/rest/uc2trustedca
3.2 https://213.205.14.13:54443/certauths/rest/uc2untrustedca
4 UC3 Smart Airport demo:
4.1 https://213.205.14.13:54443/certauths/rest/uc3trustedca
4.2 https://213.205.14.13:54443/certauths/rest/uc3untrustedca

The endpoints accepts both HTTP GET and POST methods. The GET method returns a simple text message intended to provide a visual check on the status of the service:



Alternatively, we could use curl to 'GET' the endpoint:



Use the HTTP POST method to generate the required credential/s. The trusted CAs issue X.509 certificates for trusted mF2C infrastructure components, e.g. CAU. The client should provide a max 64 chars length common-name in plain text to the appropriate trusted CA endpoint which should return the X.509 certificate and the associate private key in pem format. Users should obtain the credentials out-of-band as they are expected to be distributed with the components.
Figure 1 gives an example on obtaining the credentials using curl with the endpoint http://213.205.14.13:8080/certauths/rest/it2trustedca and the common-name 148.62.113.142. Note that the private key and X.509 certificates are concatenated together and users should use a text editor to break them into two files.

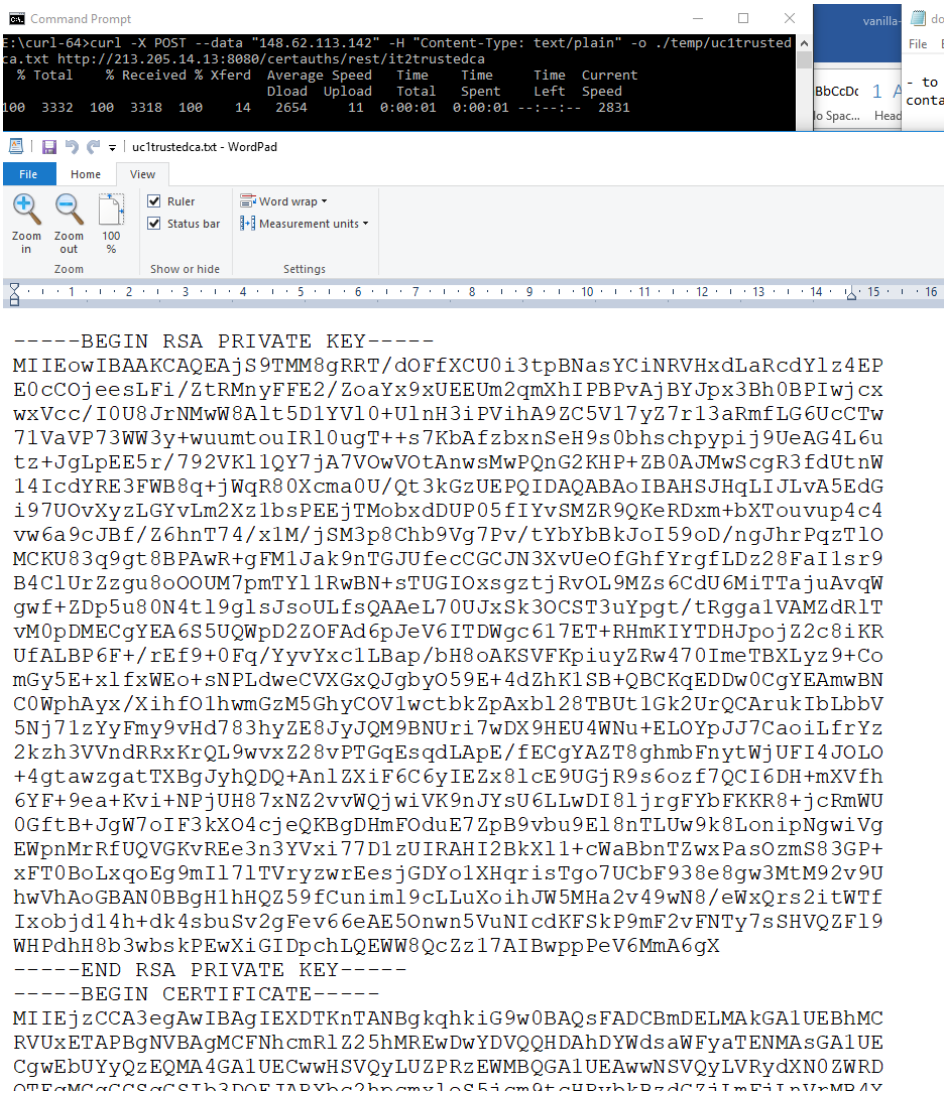


Figure 1 Using curl to obtain trusted credentials

Figure 2 shows an example using Google Advanced Rest Client (ARC) tool to obtain an X.509 certificate from an untrusted CA. The client is expected to send an PKCS10 Certification Request String in plain text. Untrusted CAs issue certificates to mF2C agents and the process takes place programmatically during the Discovery, Authentication (Security) and Categorization process flow (see D5.1).

The application provides a Junit (CertAuthRestServiceTest.java) which gives programmatic examples on calling the trusted and untrusted CAs. A Python client example (CAClientExample.py) is provided in the src/main/resources/ folder.

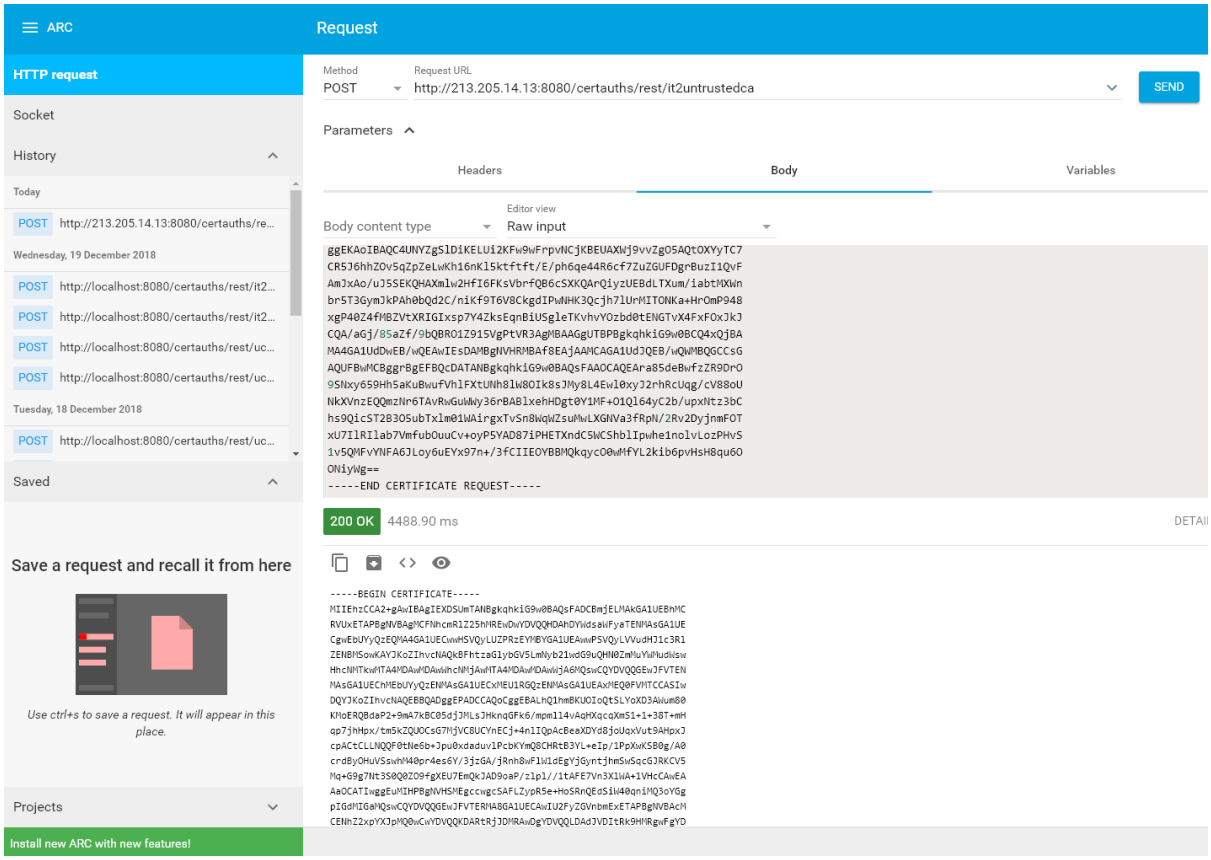


Figure 2 ARC example on getting a certificate from the trusted CA