SMARTBEAR

CALL H2020-SC1-FA-DTS-2018-2020
Trusted digital solutions and Cybersecurity in Health and Care
TOPIC DT-TDS-01-2019
Smart and healthy living at home

SMART BEAR

"Smart Big Data Platform to Offer Evidence-based Personalised Support for Healthy and Independent Living at Home"

D5.1 – Continuous Security Assurance & Privacy by design - enabling mechanisms v1
Demonstrator

Due date of deliverable: 31/01/2021
Actual submission date: 4/02/2021

Grant agreement number: 857172
Start date of project: 01/09/2019
Revision

Lead contractor: CNR
Duration: 48 months

Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020	
Dissemination Level	
PU = Public, fully open, e.g. web	✓
CO = Confidential, restricted under conditions set out in Model Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC.	
Int = Internal Working Document	

D5.1 – Continuous Security Assurance & Privacy by design - enabling mechanisms v1
Demonstrator

Editors
Ioannis Basdekis (STS)
Konstantina Koloutsou (STS)

Contributors
Othonas Soultatos (STS)
Konstantin Pozdniakov (CITY)
Fady Copty (IBM)
George Zissis (ATC)
Giorgos Giotis (ATC)

Reviewers
Manolis Michalodimitrakis (FORTH)
Maria Pateraki (FORTH)
Christos Kloukinas (CITY)

The work described in this document has been conducted within the project SMART BEAR, started in September 2019. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857172

©Copyright by the SMART BEAR Consortium.

Executive Summary

This document presents a demonstration of the security and privacy mechanisms that were developed in the first period of the Smart Bear (SB) project for the SB@Cloud.
Chapter 1 makes an introduction about the deliverable. The requirements and software prerequisites are exhibited in Chapter 2, while Chapter 3 demonstrates the installation and configurations needed to have a running system.
The objective of this demonstrator is to familiarize with the Authentication and Authorization capabilities of the software that can be used to integrate an application, in the context of SB project.

Contents

Executive Summary **4**
Contents **5**
List of acronyms **6**
List of figures **7**
1 Introduction **8**
1.1 Purpose of the document 8
2 Requirements **9**
2.1 Building Blocks 9
3 Installation - Configuration **10**
3.1 Download and install archive 10
3.2 Configuration 10
3.2.1 End-User Roles 10
3.2.2 Configuring Role-Based Access Control of Smart Bear APIs 12
3.2.3 External component registration 20
3.2.4 GDPR Requests 21
3.2.5 Log files 21
3.2.6 Alternative functionality dashboards 22
Annex A **25**
Docker-compose file 25
WSO2 identity server configuration 28
WSO2 API manager 30
Dockerfile 30
WSO2 API manager configuration 33

List of acronyms

API	Application programming interface
CPU	Central Processing Unit
Docker	an open platform for developing, shipping, and running applications
HTTP	Hyper Text Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity Access Management
IS	Identity server
JSON	JavaScript Object Notation
JWT	JSON Web Token
Kubernetes	(also known as K8s) is an open-source system for automating deployment, scaling, and management of containerized applications
Linux	a family of open-source Unix-like operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds
OAuth	open standard for access delegation
RAM	Random Access Memory
RBAC	Role Based Access Control
REST API	(also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services
SB	Smart Bear
SB@App	Smart Bear smartphone app
SB@Cloud	Smart Bear Cloud infrastructure
SB@Dashboard	Smart Bear Dashboard component
SB@HomeHub	Smart Bear Home Hub component
SB@Repository	Smart Bear Repository component
SB@SecurityComponent	Smart Bear Security component
SDK	Software Development Kit
TLS	Transport Layer Security
UML	Unified Modelling Language
X.509	X.509 is a standard defining the format of public key certificates
XACML	eXtensible Access Control Mark-up Language
XML	eXtensible Mark-up Language

List of figures

Figure 1: Software has been downloaded successfully and is running 10
Figure 2: Step 1/3: Identity Server login 11
Figure 3: Step 2/3: Role addition 11
Figure 4: Step 3/3: Roles added 12
Figure 5: API manager login 13
Figure 6: Scope creation 13
Figure 7: Scope role binding 14
Figure 8: New API creation 15
Figure 9: New API definition 15
Figure 10: New API overview 16
Figure 11: Resource addition 16
Figure 12: rest endpoint creation and scope binding 17
Figure 13: Message mediation (end-user triggers this clicking on the red colour-coded pen icon) 17
Figure 14: User data pseudonymisation mediation 18
Figure 15: Data pseudonymisation example 18
Figure 16: API overview (end-user triggers this clicking on the red colour coded Publish btn) 19
Figure 17: API published (updated status indicated in the red colour-coded area) 20
Figure 18: App subscription and credentials 21
Figure 19: WSO2 identity server user portal 23
Figure 20: WSO2 API manager admin page 24
Figure 21: API rate limiting 24

1 Introduction

1.1 *Purpose of the document*

This deliverable describes the current state of the Smart Bear (SB) Security Component (SB@SecurityComponent) implementation, a component that resides in the SB@Cloud and consists of the following two sub-components: a) the Secure Manager One and b) the Secure User Data Storage.

In particular, the following captures installation and configuration settings that are linked to the SB architecture (presented in D2.2), and shows how the Secure One Manager used to support the Role-Based Access Control (RBAC), how a unique session in this context is generated, the mechanism responsible for removing and replacing Ids to preserve the data privacy, and other security related features.

The SB architecture follows a modern hybrid-cloud model, where part of the services is deployed on private cloud, and another part is deployed on a public cloud (presented in D2.2). To secure this hybrid-cloud model there is a need for a solution (the SB@SecurityComponent) that provides security services for such a hybrid cloud model. As an initial solution and for the purpose of this Demonstrator, the underneath layer upon which SB@SecurityComponent is deployed, uses Kubernetes for container orchestration to cope with the deployment, management, scaling, and networking of containers across different environments without needing to redesign it. Current version also uses OpenShift (OpenShift is a cloud-based Kubernetes container platform that's considered both containerization software and a platform-as-a-service, offering security, built-in monitoring, centralized policy management, and compatibility with Kubernetes container workloads), an open source on-premises platform as a service built around Docker containers that are orchestrated and managed by Kubernetes, to provide all the security capabilities required for platform security and augments the application security capabilities provided by Kubernetes. As such, the following sections present results of the initial phase of work on the SB@SecurityComponent, while the first stage of integration is about to be completed. In this respect, references might be changed to the appropriate domain that the SB@Cloud components will run.

Pending action: The source code and associated configuration files presented in Annex A (and all future updates) will be uploaded to the SB@Repository to be available to download.

2 Requirements

2.1 *Building Blocks*

The system requirements mentioned below are requirements to run the SB@SecurityComponent for the specified load of SB (in D2.1).

Hardware requirements:

- RAM: The minimum RAM proposed is 4GB
- CPU: the recommended number of cores is 2
- Disk: For the Secure Manager One, 20 GB of storage is sufficient

Software:

- Operating system: It is recommended to run the installation on a Linux operating system
- Docker: Docker is used to create the components that are needed to run. The official website of Docker provides detailed installation instructions[1]
- Docker compose: The deployment of SB@SecurityComponent is based on a Docker compose file for easier deployment and management. The official website of Docker provides detailed installation instructions[2]

A stable internet connection is required in the initial deployment, since the docker-compose needs to download software from global repositories.

Pending action: Final version of the integrated framework will be reported in D6.2.

[1] https://docs.docker.com/engine/install/

[2] https://docs.docker.com/compose/install/

3 Installation - Configuration

3.1 *Download and install archive*

The system administrator:

- After downloading and extracting the archive, navigates inside directory named “secure manager one”, where the docker-compose.yaml file resides.
- (windows terminal windows) Executes the command “docker-compose up –build”. This command will download all the necessary files and create the necessary for this demo containers.

Depending on network speed, downloading may take a while to complete. Informative messages will appear about successful installation of included services during the installation process (depicted in Figure 1).

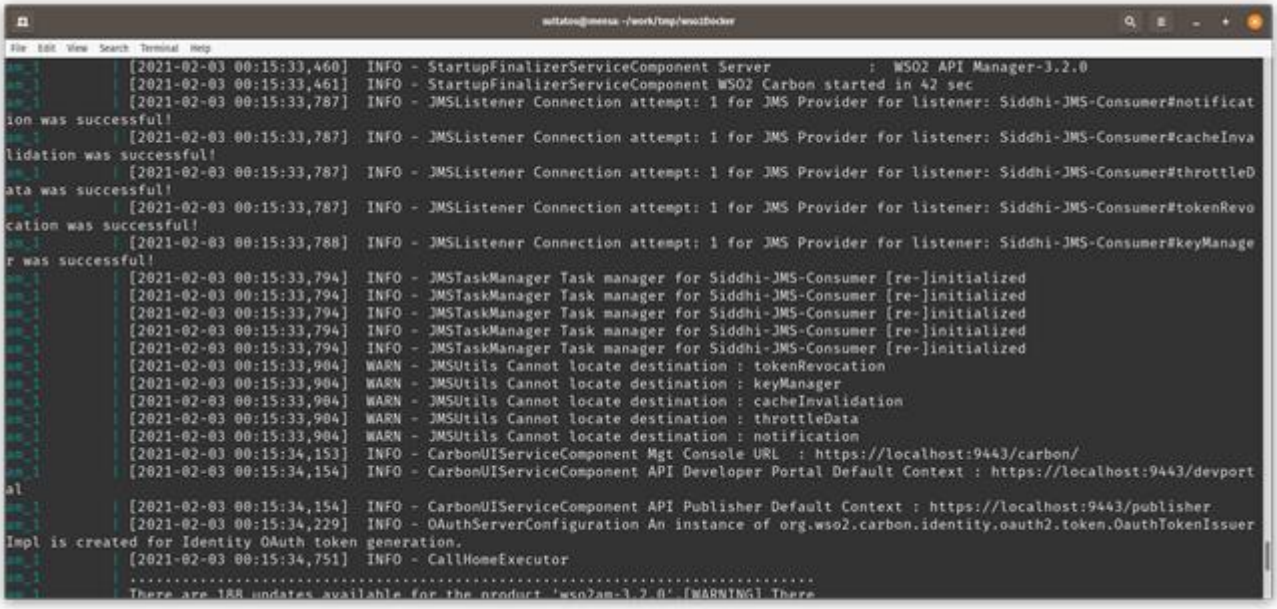

Figure 1: Software has been downloaded successfully and is running

This output confirms that services are running, and then one can proceed to the next stage of the installation and configuration process.

Pending action: SB@Repository will provide means and instructions for downloading SB project’s archives.

3.2 *Configuration*

3.2.1 End-User Roles

The following images (Figure 2 and Figure 3) depict the process by which SB roles (defined in D2.1 and D5.2) were created. To create a role the system administrator:

1. Logs in at the identity server
2. From the menu (on the left) triggers action *Add a new role*

3. Inserts the role's name and selects the permissions. The role's permissions can be overridden when assigning role to the end-user
4. Repeats steps 2 and 3 until all roles are created (Figure 4)

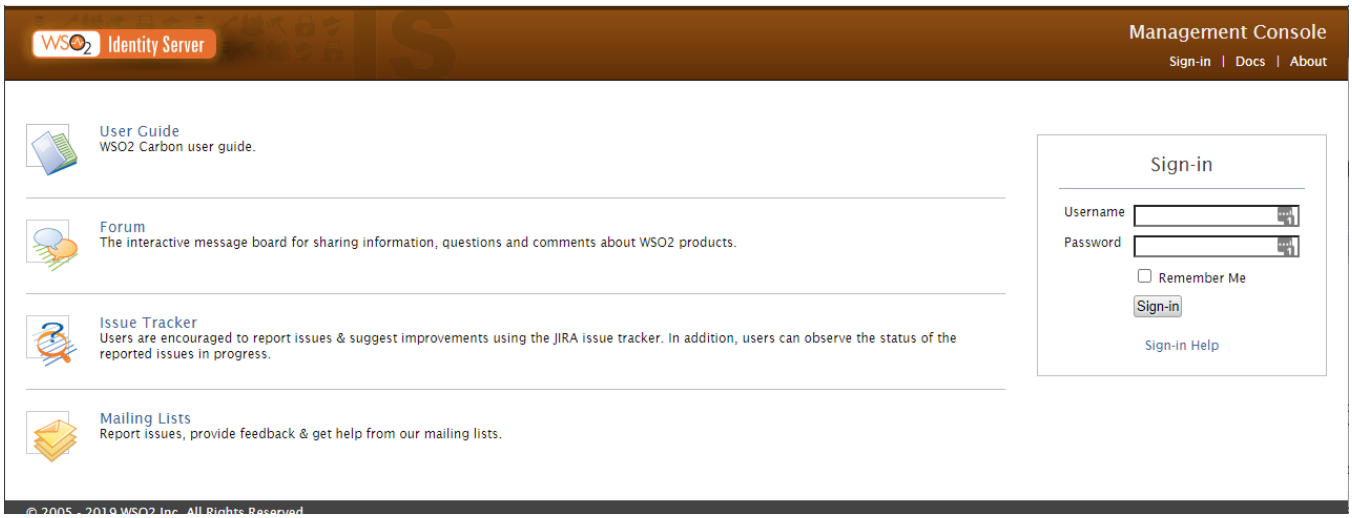

Figure 2: Step 1/3: Identity Server login

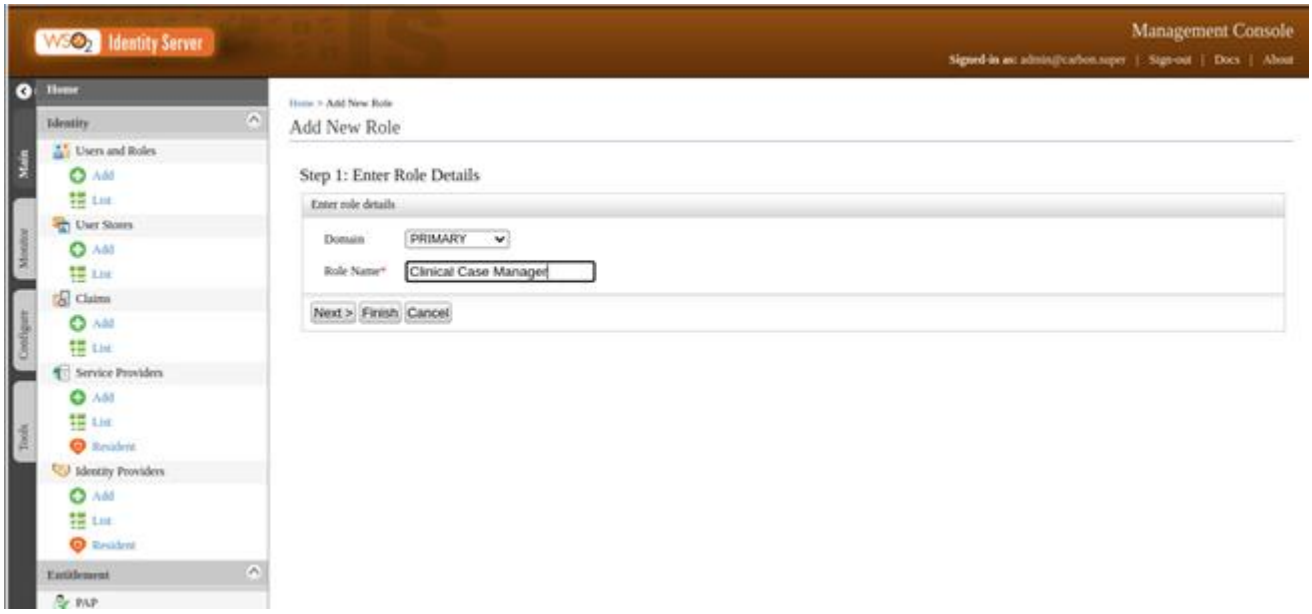

Figure 3: Step 2/3: Role addition

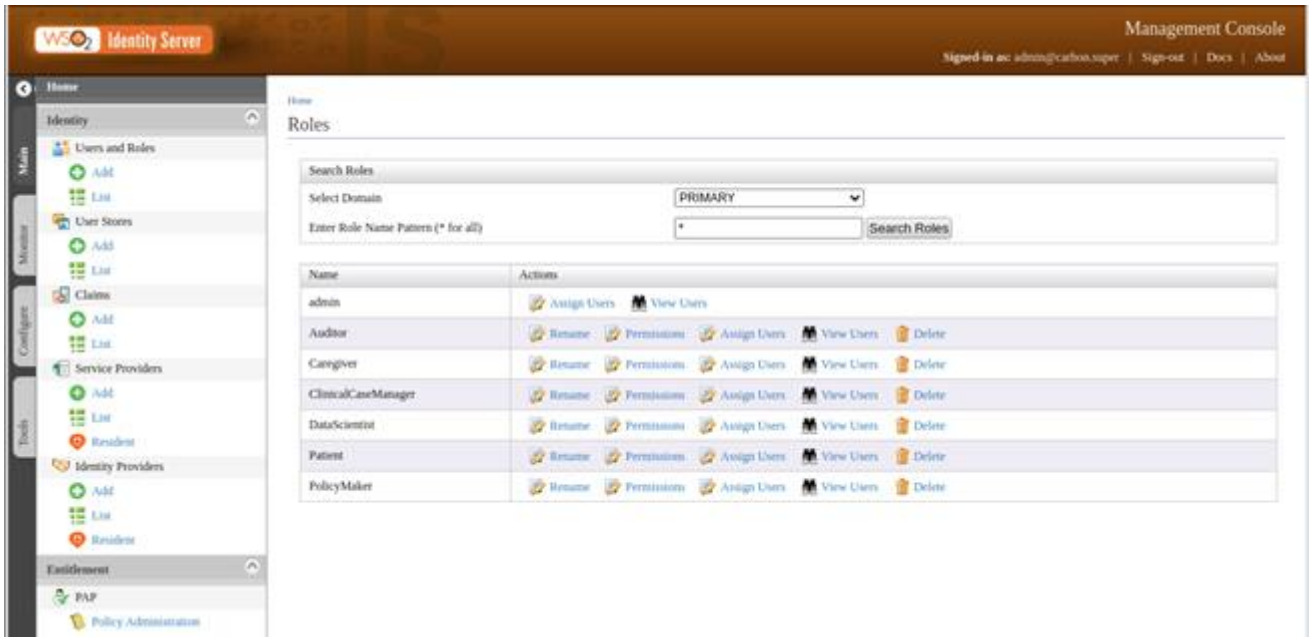

Figure 4: Step 3/3: Roles added

3.2.2 Configuring Role-Based Access Control of Smart Bear APIs

Within SB@Cloud, a REST layer is used for retrieving, saving or analysing the data stored in the SB@Repository. In order to allow this type of access, system administrator creates the specific REST endpoints that the Secure Manager One will expose to the different SB@Cloud components and the SB@App.

1. From https://localhost:9443/publisher, the browser will redirect system administrator to the login page (Figure 5).
2. The administrator may proceed in creating scopes (e.g., Figure 6) used for SB REST APIs. Scopes are a way to bind user-roles to APIs.
3. To create a scope, after the login page seen in Figure 5, from the top navigation bar system administrator clicks on the *Scopes* menu tab (Figure 6). Then after pressing the *Create new scope* button, a creation form wizard will appear (Figure 7).

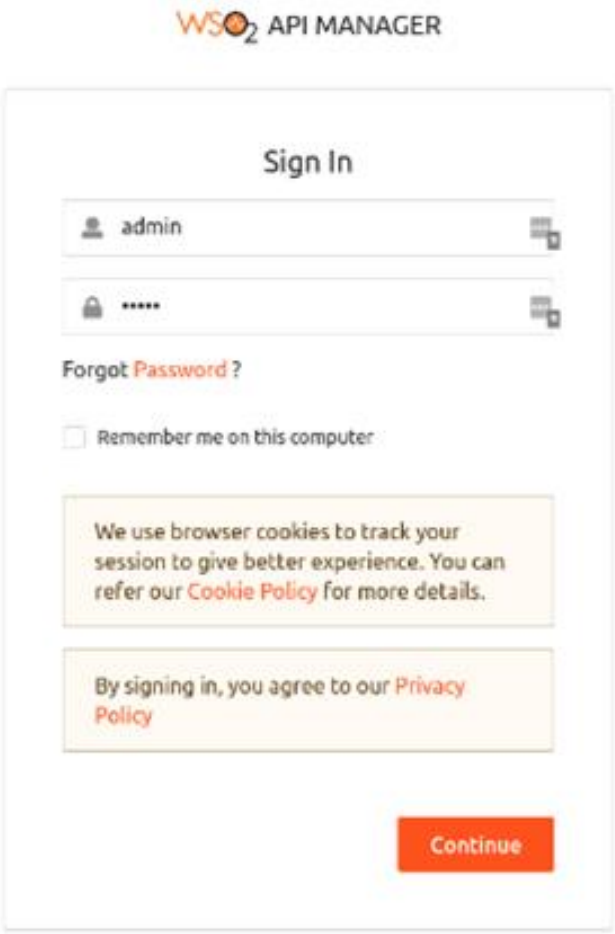

Figure 5: API manager login

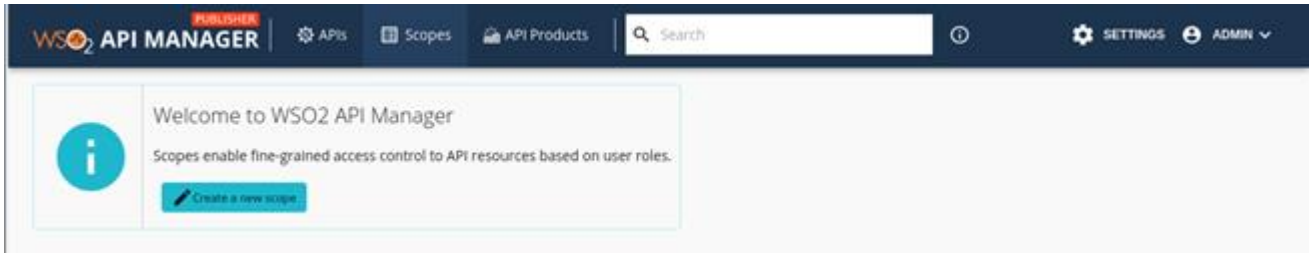

Figure 6: Scope creation

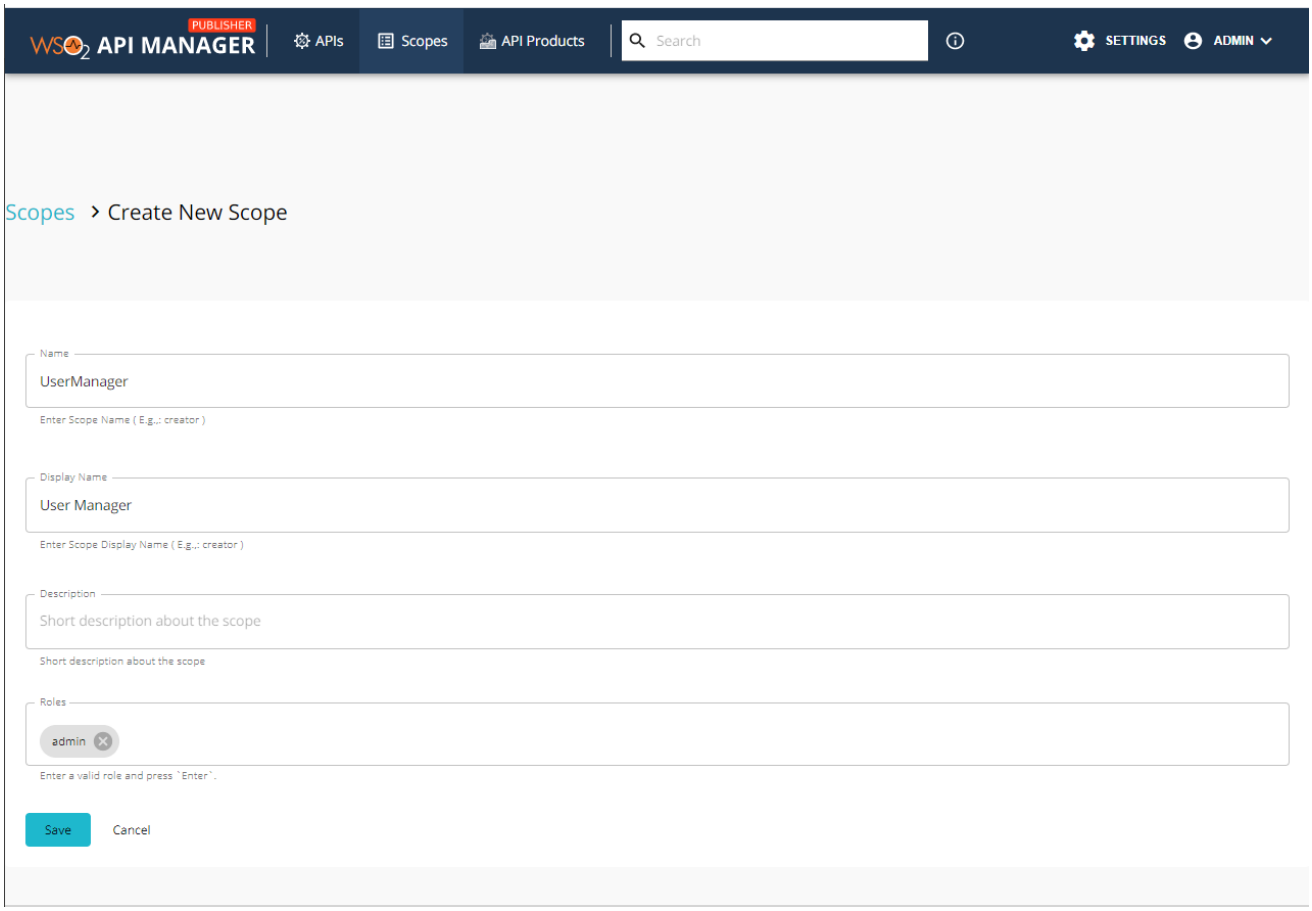

Figure 7: Scope role binding

By using this wizard (e.g., as seen in Figure 7), the system administrator binds that the scope userManager is used with the role admin:

- Scope Name: A unique key for identifying the scope.
- Scope Display Name: A human-readable name for the scope.
- Scope Description: The description for the scope.
- Roles: The user role/roles that are allowed to obtain a token against this scope.

4. The administrator fills the details required and presses the *Save* button.
5. Later on, the administrator proceeds with the creation of a new API definition by navigating from the top menu bar to the APIs navigation menu and presses the *Design a New REST API* button (Figure 8). The administrator fills all details required such as: name, version, context and endpoint:
    - Name: A human-readable name for the API.
    - Version: API version
    - Context: API's root context when invoking the API through the Gateway. It is used by the Gateway to identify the API.
    - Endpoint: a link that will be used as an endpoint for the API.

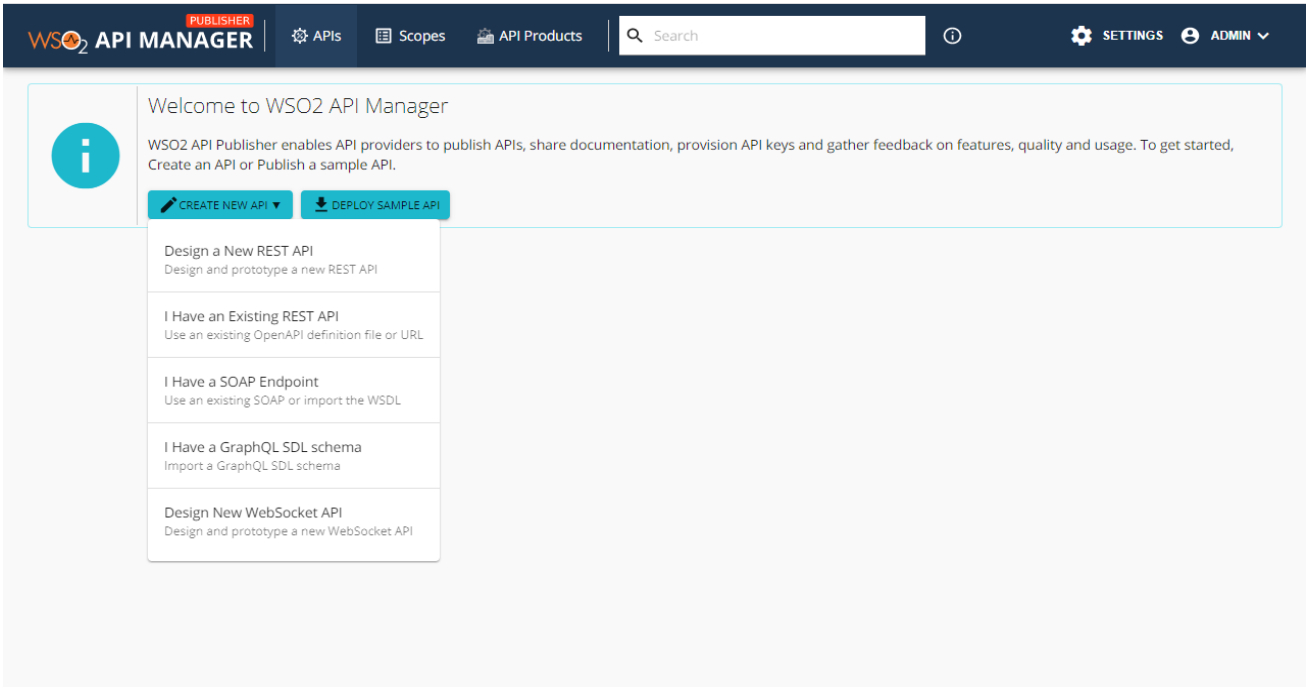

Figure 8: New API creation

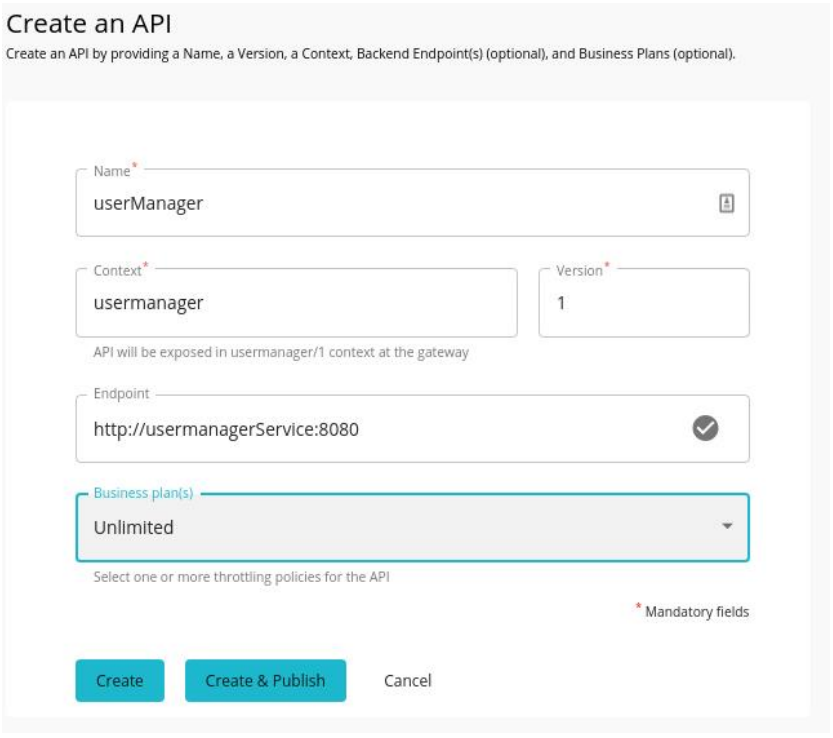

Figure 9: New API definition

6. By pressing the *Create* button, a new API service will be created (as seen in Figure 9. The newly created service definition of the selected service is presented (Figure 10).

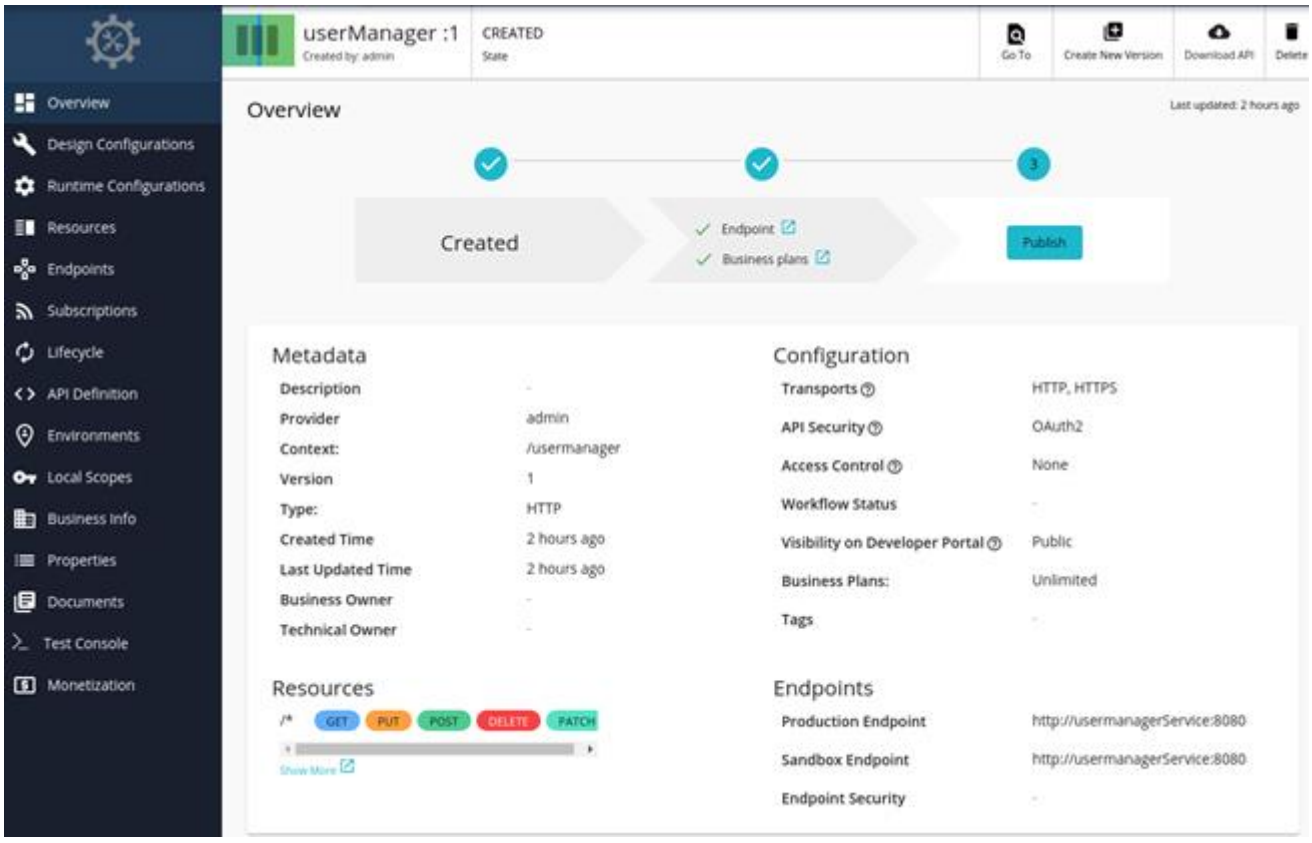

Figure 10: New API overview

In order to add in the API additional endpoints, the system administrator navigates to the resources tab and presses the blue plus (+) button (as presented in Figure 11), for every REST endpoint he/she wants to add.

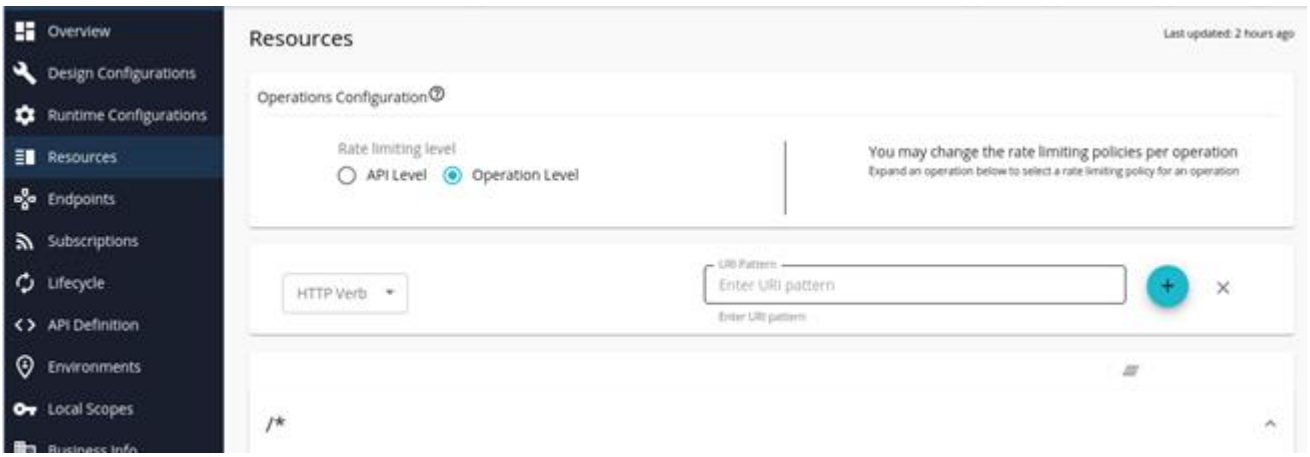

Figure 11: Resource addition

The following figure (Figure 12) depicts that the newly created /user rest endpoint can only be invoked with the scope of the userManager. As indicated, this scope is available only to end-users to whom the admin role has been assigned.

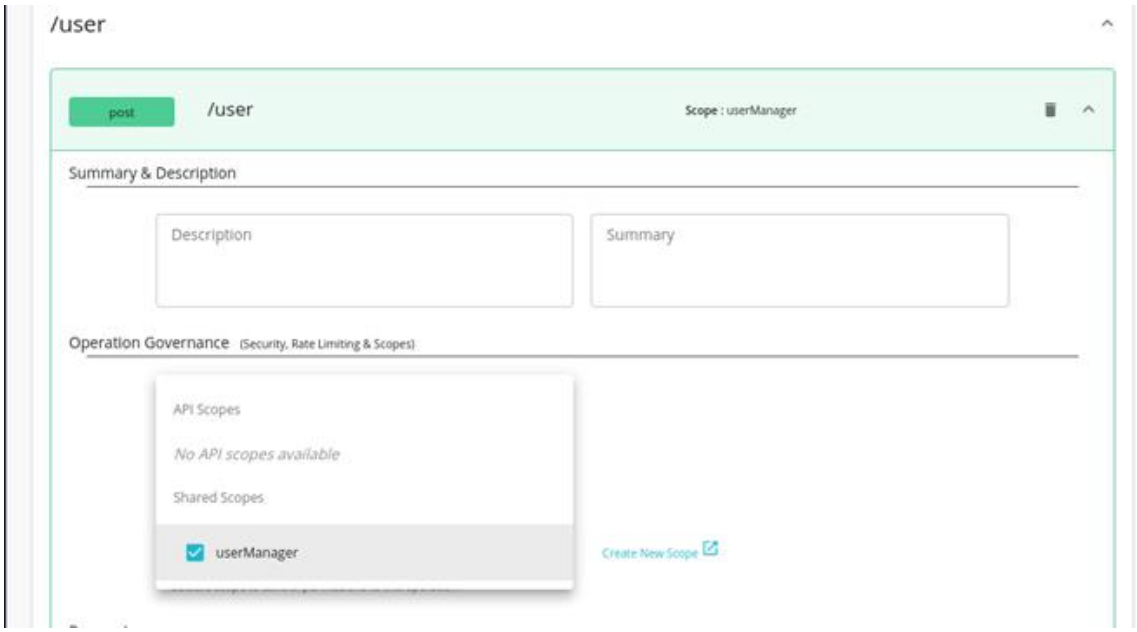

Figure 12: rest endpoint creation and scope binding

After finishing the API specification, the system administrator navigates at the left menu to the Runtime Configurations screen to define the message mediators for this specific endpoint. The message mediators are prebuilt software that must be imported in order to support the SB data pseudonymisation mechanism (presented in D5.2).

Message mediators are software that are called during each endpoint request and can alter the data of this request prior to forward this request to the backend service. To add a Message Mediation, the system administrator clicks on the Pen icon (as shown in Figure 13), and then adds the mediator by using the Mediation Policy form (Figure 14). Mediator is represented by an XML file containing mediation policy which will be uploaded after the select button will be pressed.

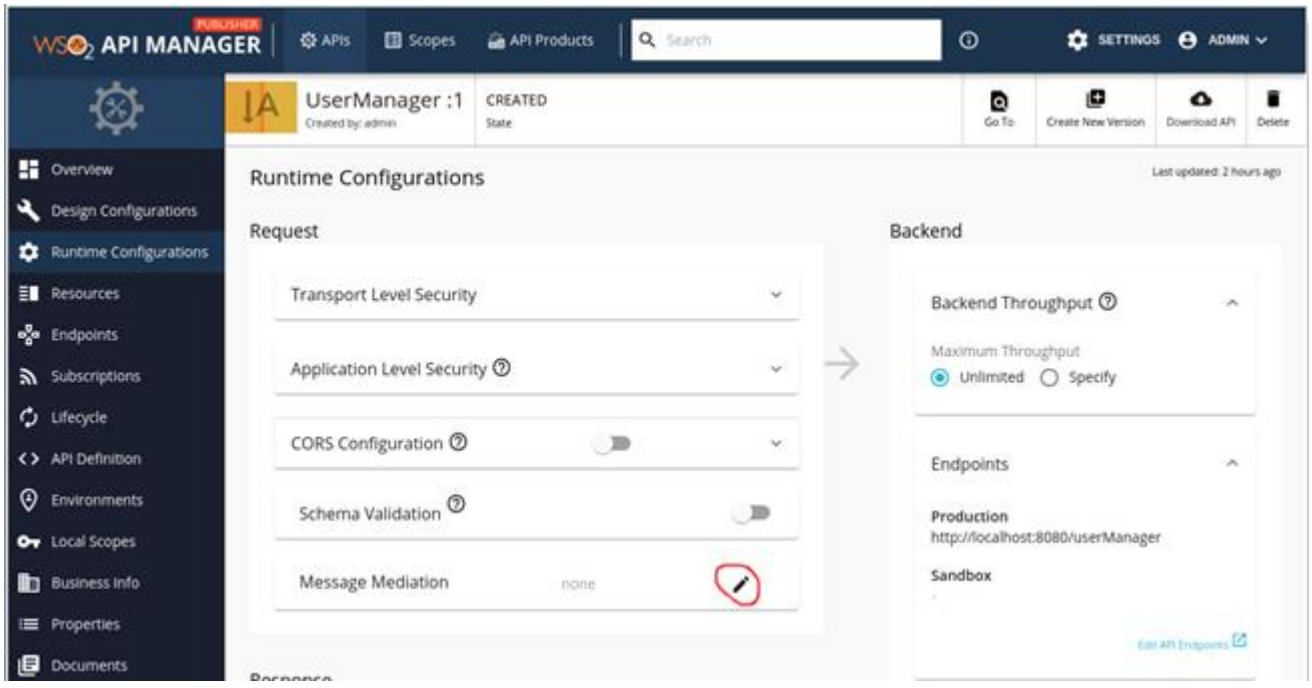

Figure 13: Message mediation (end-user triggers this clicking on the red colour-coded pen icon)

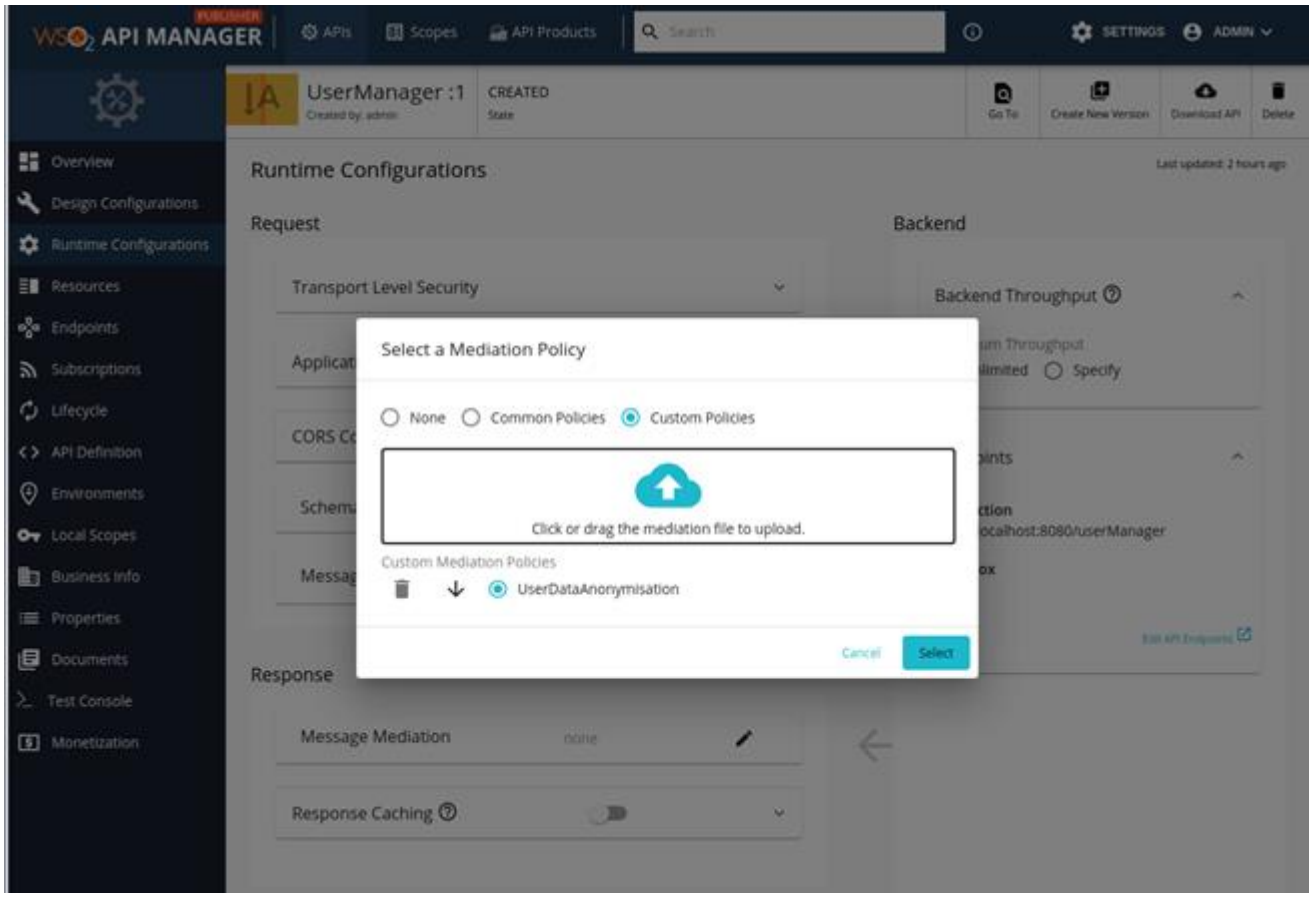

Figure 14: User data pseudonymisation mediation

The following example (Figure 15) illustrates (on the right) a FHIR JSON that contains patient's information, and (on the left) the associated user profile created by the system (note: a red color-coded indicator marks the association made between the data). The pseudonymisation mediation will strip those data and replace them with the SB Pseudo-Id2 to preserve the anonymity (Ids replacement mechanism presented in detail in D5.2).

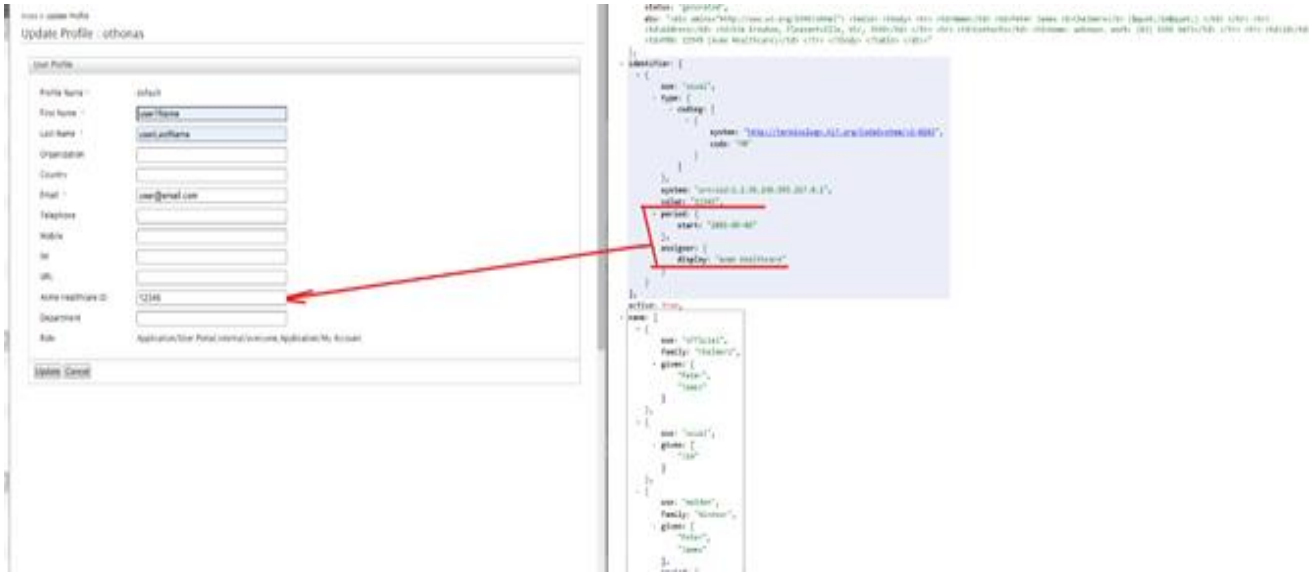

Figure 15: Data pseudonymisation example

Upon creation completion, the system administrator may examine the details of this configuration (e.g., metadata, associated endpoints, and other information) by navigating to *Overview* tab (Figure 16), and clicking on the Publish button to publish this API. In case the API will not be in "published" state, admin can go to Lifecycle tab and click "redeploy" to re-publish the API.

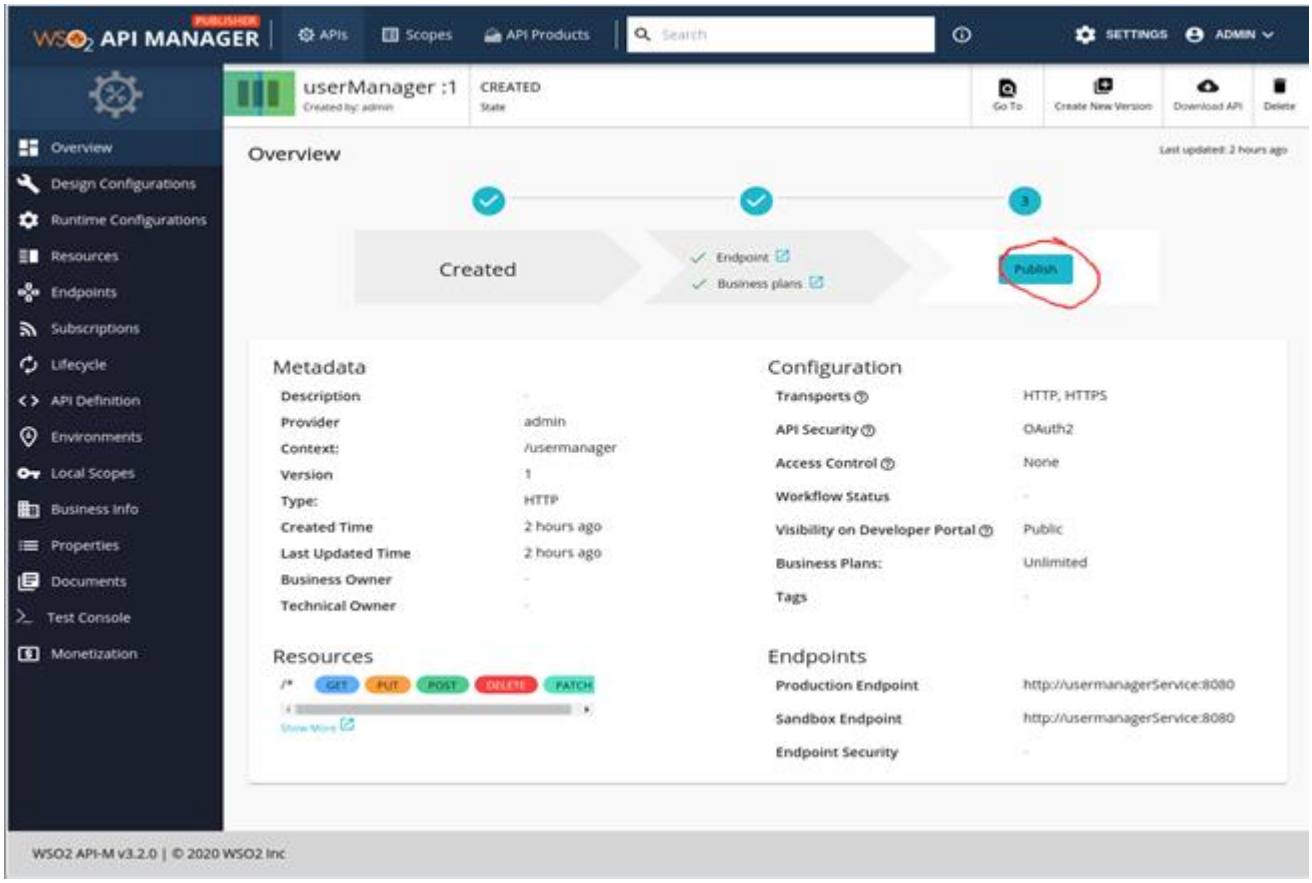

Figure 16: API overview (end-user triggers this clicking on the red colour coded Publish btn)

Upon successful completion, they click on the link *View in developer portal* (Figure 17) which redirects to the developer's portal. This system is used by the developers in order to register their applications to consume the specified APIs.

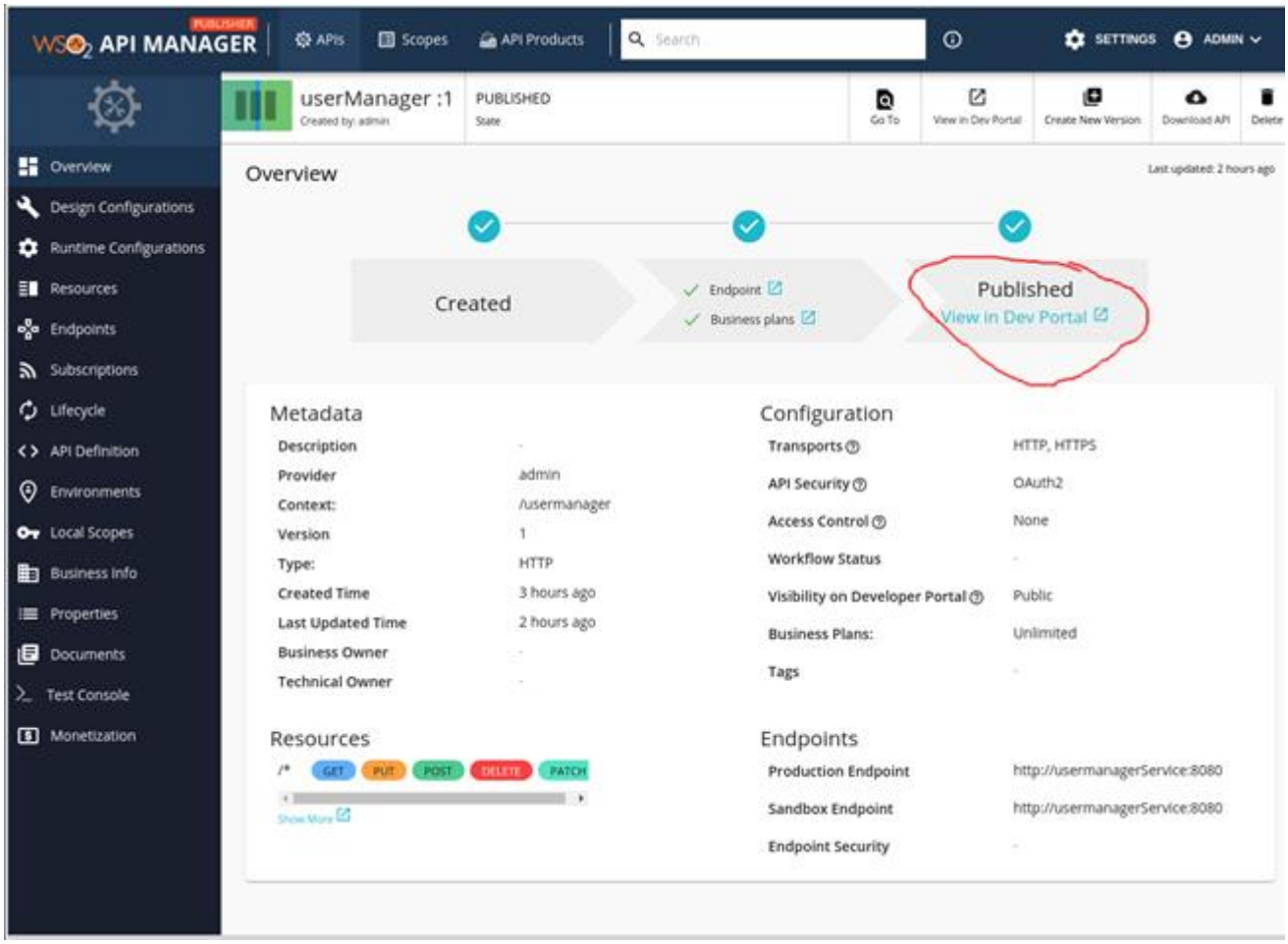

Figure 17: API published (updated status indicated in the red colour-coded area)

3.2.3 External component registration

To be able to consume the APIs, a developer needs to initially register the application in the developer's portal. Initially, they must create an application from the menu. To do so the developer:

1. navigates to subscriptions on the left navigation bar (Figure 18)
2. subscribes, then presses production keys and then generates keys. Those will be later used by the SB@App for example.

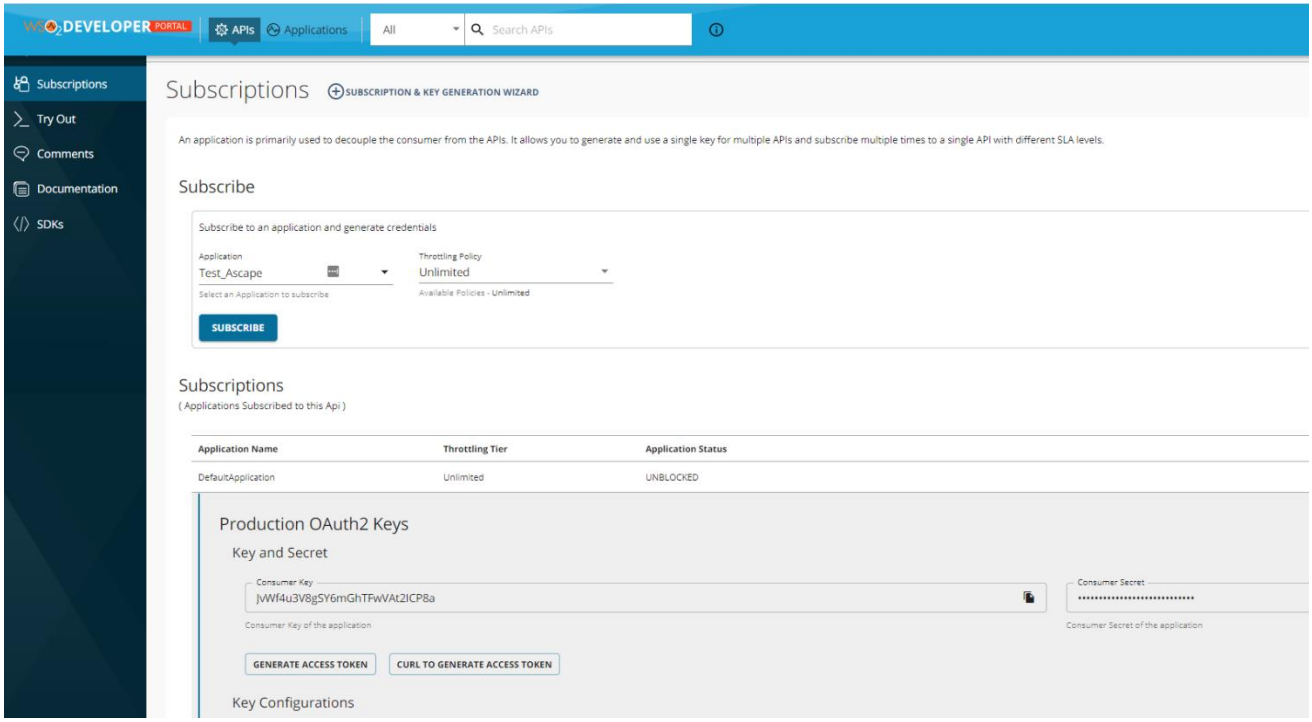

Figure 18: App subscription and credentials

Pending action: In all examples demonstrated (and associated configurations in Annex), the domain name used is "localhost". In a final production environment these references will be changed to indicate the final SB domain in which SB components will run. In addition, the default admin password should be changed as well (by altering the configuration file notation).

3.2.4 GDPR Requests

Pending action: SB@Dashboard interfaces and database schema associated to the SB@SecurityComponent REST functionality presented in D5.2 (section 3.2 SB@SecurityComponent: Core Component Specification), will be integrated upon completion of D4.2, and will be reported in D6.2.

3.2.5 Log files

The secure manager one consists of off-the-shelf components that have by default logging capabilities that are useful for monitoring the status of the system. Also, those logs can be used to audit a GDPR request.

Various log types that are used in WSO2 products are listed below.

- **Carbon logs**: All WSO2 products generate administrative and server-side logs by utilizing log4j [3]logging capabilities
- **Audit logs**: Audit logs are used for tracking the sequence of actions that affect a particular task carried out on the server.

[3] https://logging.apache.org/log4j/2.x/

- **HTTP access logs**: HTTP requests/responses are logged in access logs to monitor the activities related to an application's usage.

Log growth can be managed by the following configurations in the log4j configuration file. The log rotation[4] of the files is configurable. By default, log rotation is daily. Log rotation based on time as opposed to size is the default choice because it helps to inspect the events that occurred during a specific time. Log files are archived to maximize the use of space.

3.2.5.1 Smart Bear use case

In the context of Smart Bear project, the WSO2 Identity Server carbon and Audit logs will be utilized to analyze the events that happened in the user data, e.g. who updated the contact info of a person. The WSO2 API manager Audit and HTTP access logs can be utilized to have a general knowledge of the endpoints that each user invokes for traceability and future system optimizations.

3.2.6 Alternative functionality dashboards

Since WSO2 components are off the shelf components, they can provide the administrator with more complicated functionalities. Below are some screenshots depicting those functionalities. Figure 19 shows a user portal that can be alternatively used by the users to edit their information, setup account recovery info and also setup multi-factor authentication. Figure 20 presents the administration page of all the REST endpoints, and also other metadata that may be useful to the Smart Bear administrators. Another feature that can be leveraged by admin is the ability to create or edit rate limiting policies as seen in Figure 21 that then can be applied to API endpoints.

[4] https://en.wikipedia.org/wiki/Log_rotation

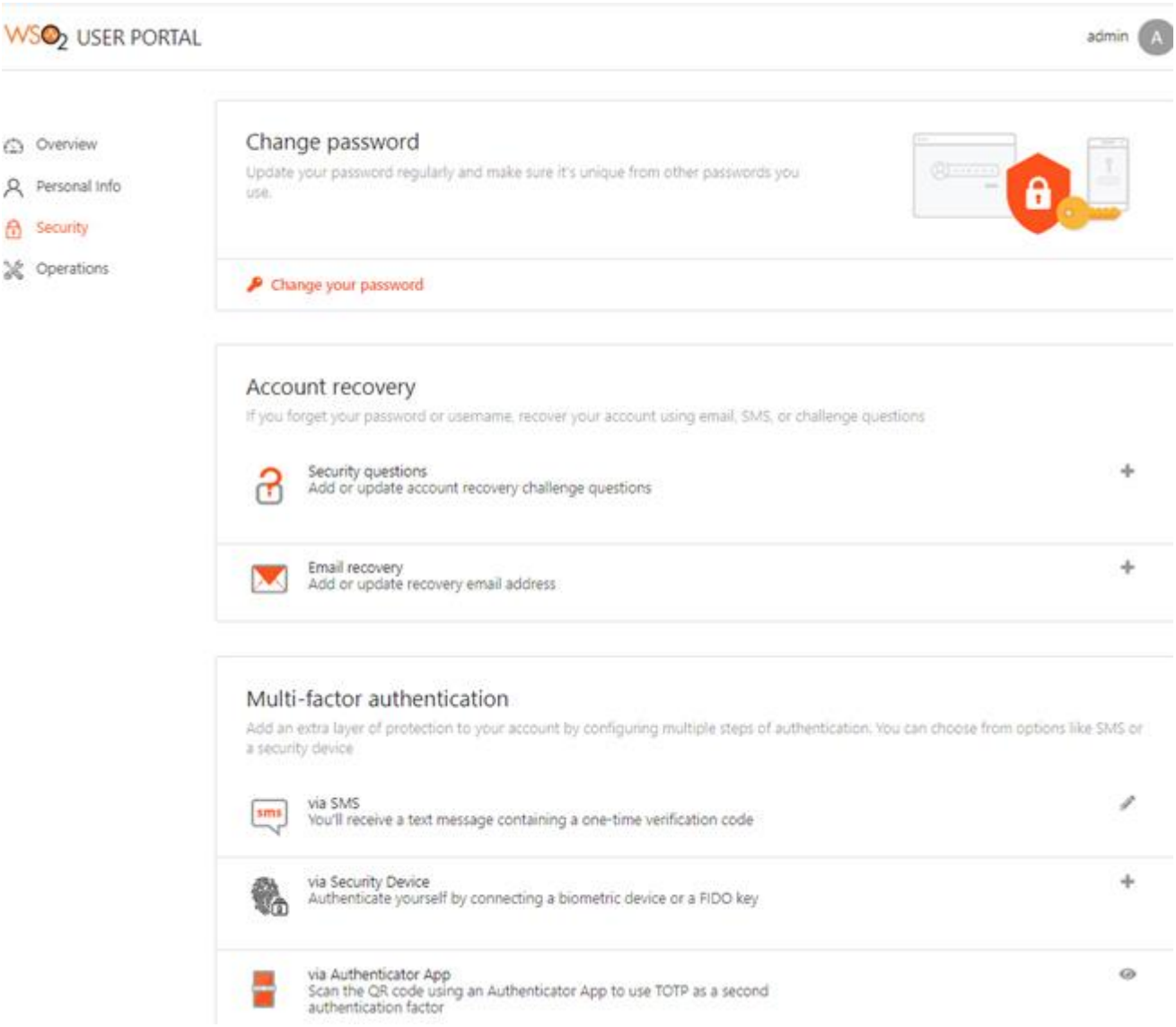

Figure 19: WSO2 identity server user portal

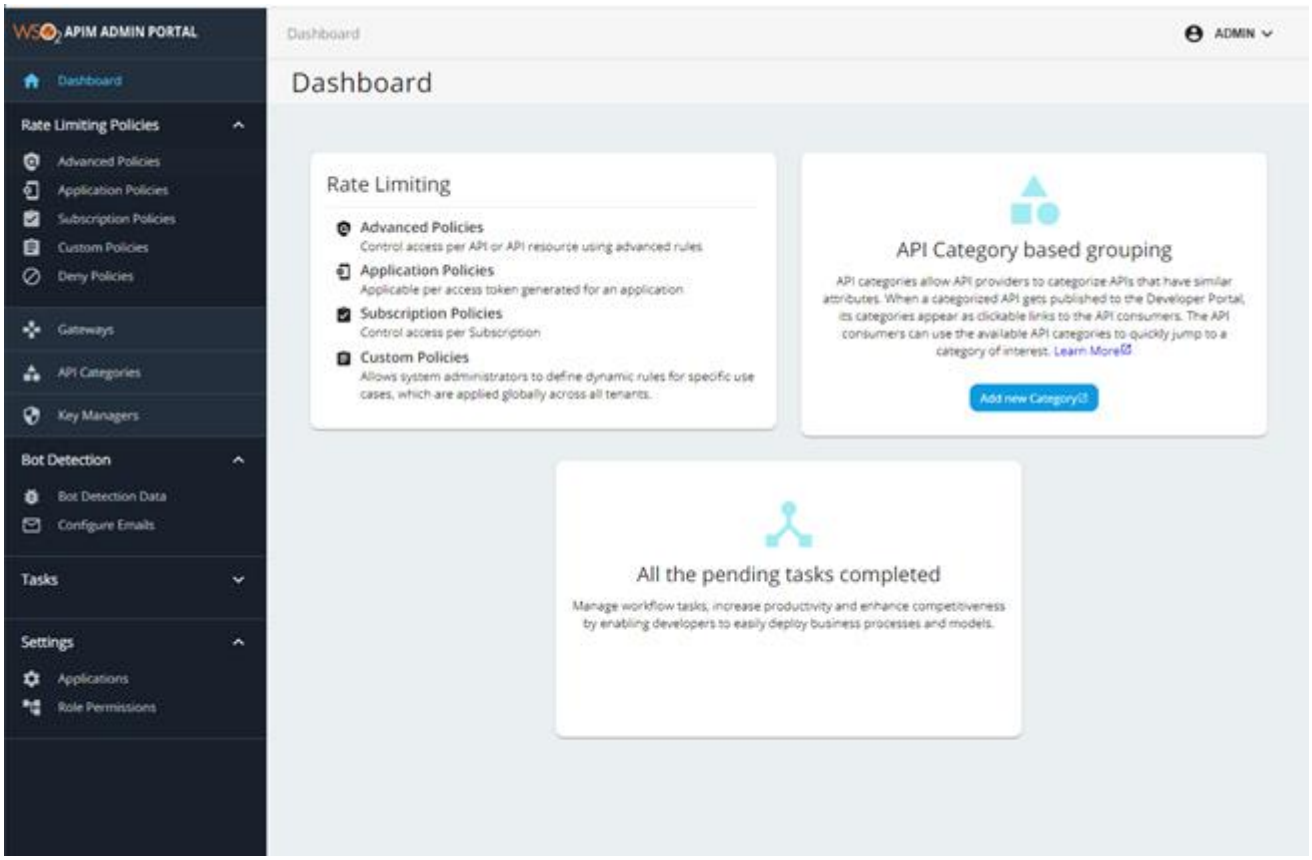

Figure 20: WSO2 API manager admin page

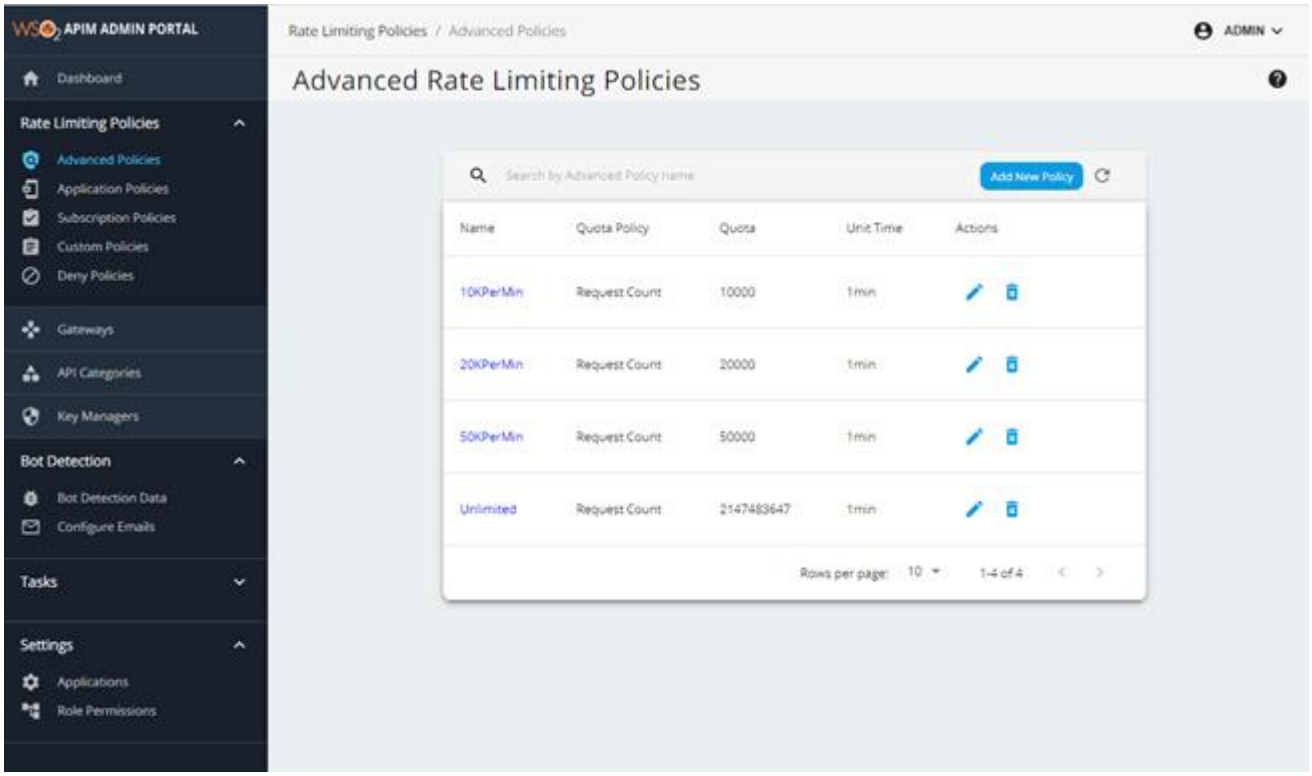

Figure 21: API rate limiting

Annex A

Docker-compose file

```
version: '3'
services:
  is:
    build: ./is/
    healthcheck:
      test: ["CMD", "nc", "-z","localhost", "9443"]
      interval: 10s
      start_period: 180s
      retries: 20
    depends_on:
     postgres:
        condition: service_healthy
    ports:
      - "9444:9443"
    volumes:
     - isdata:/opt/wso2is-5.10.0/repository/
  am:
    build: ./am/
    healthcheck:
      test: ["CMD", "nc", "-z","localhost", "9443"]
      interval: 10s
      start_period: 180s
      retries: 20
    depends_on:
      postgres:
        condition: service_healthy
      is:
         condition: service_healthy
    ports:
      - "9443:9443"
      - "8280:8280"
      - "8243:8243"
    volumes:
     - amdata:/opt/wso2am-3.2.0/repository/
  postgres:
       build: ./postgres/
       restart: always
       healthcheck:
         test: ["CMD-SHELL", "pg_isready -U regadmin"]
```

```
        interval: 10s
        timeout: 5s
        retries: 5
      volumes:
        - postgresdata:/var/lib/postgresql/data
volumes:
  postgresdata:
  isdata:
  amdata:
```

Security Component database

Dockerfile

```
FROM postgres:13-alpine
ENV POSTGRES_USER=regadmin
ENV POSTGRES_PASSWORD=regadmin
COPY scripts/docker_postgres_init.sql /docker-entrypoint-initdb.d/
COPY scripts/postgresql_shared.sql /docker-entrypoint-initdb.d/
COPY scripts/postgresql_apim.sql /docker-entrypoint-initdb.d/
EXPOSE 5432
```

WSO2 Identity Server

Dockerfile

```
FROM adoptopenjdk/openjdk11:jdk-11.0.8_10-alpine
ARG USER=wso2carbon
ARG USER_ID=802
ARG USER_GROUP=wso2
ARG USER_GROUP_ID=802
ARG USER_HOME=/home/${USER}
# build arguments for WSO2 product installation
ARG WSO2_SERVER_NAME=wso2is
ARG WSO2_SERVER_VERSION=5.10.0
ARG WSO2_SERVER_REPOSITORY=product-is
ARG WSO2_SERVER=${WSO2_SERVER_NAME}-${WSO2_SERVER_VERSION}
ARG WSO2_SERVER_HOME=${USER_HOME}/${WSO2_SERVER}
ARG
WSO2_SERVER_DIST_URL=https://github.com/wso2/${WSO2_SERVER_REPOSITORY}/releas
es/download/v${WSO2_SERVER_VERSION}/${WSO2_SERVER}.zip
# build arguments for external artifacts
ARG DNS_JAVA_VERSION=2.1.8
ARG K8S_MEMBERSHIP_SCHEME_VERSION=1.0.7
ENV ENV=${USER_HOME}"/.ashrc"

# create the non-root user and group and set MOTD login message
RUN \
    addgroup -S -g ${USER_GROUP_ID} ${USER_GROUP} \
```

```
    && adduser -S -u ${USER_ID} -h ${USER_HOME} -G ${USER_GROUP} ${USER} \
    && echo ${MOTD} > "${ENV}"

# create Java prefs dir
# this is to avoid warning logs printed by FileSystemPreferences class
RUN \
    mkdir -p ${USER_HOME}/.java/.systemPrefs \
    && mkdir -p ${USER_HOME}/.java/.userPrefs \
    && chmod -R 755 ${USER_HOME}/.java \
    && chown -R ${USER}:${USER_GROUP} ${USER_HOME}/.java

# install required packages
RUN apk add --no-cache netcat-openbsd
# RUN mkdir -p ${WSO2_SERVER_HOME}
# COPY  wso2is/ ${WSO2_SERVER_HOME}/
# RUN chown wso2carbon:wso2 -R ${WSO2_SERVER_HOME}

# add the WSO2 product distribution to user's home directory
RUN \
   wget --no-check-certificate -O ${WSO2_SERVER}.zip "${WSO2_SERVER_DIST_URL}" \
   && unzip -d ${USER_HOME} ${WSO2_SERVER}.zip \
   && chown wso2carbon:wso2 -R ${WSO2_SERVER_HOME} \
   && rm -f ${WSO2_SERVER}.zip

# add libraries for Kubernetes membership scheme-based clustering
ADD                                                    --chown=wso2carbon:wso2
https://repo1.maven.org/maven2/dnsjava/dnsjava/${DNS_JAVA_VERSION}/dnsjava-
${DNS_JAVA_VERSION}.jar ${WSO2_SERVER_HOME}/repository/components/lib
ADD                                                    --chown=wso2carbon:wso2
http://maven.wso2.org/nexus/content/repositories/releases/org/wso2/carbon/kubernetes/artifacts/kube
rnetes-membership-scheme/${K8S_MEMBERSHIP_SCHEME_VERSION}/kubernetes-
membership-scheme-${K8S_MEMBERSHIP_SCHEME_VERSION}.jar
${WSO2_SERVER_HOME}/repository/components/dropins

# copy extensions to the identity server home
COPY                          --chown=wso2carbon:wso2                  files/dropins
${WSO2_SERVER_HOME}/repository/components/dropins/
# copy customized webapps to the identity server home
COPY                          --chown=wso2carbon:wso2                  files/webapps
${WSO2_SERVER_HOME}/repository/deployment/server/webapps/

# set the user and work directory
COPY --chown=wso2carbon:wso2  init.sh ${USER_HOME}/
USER ${USER_ID}
```

```
WORKDIR ${USER_HOME}

COPY --chown=wso2carbon:wso2 deployment.toml ${WSO2_SERVER_HOME}/repository/conf/
# set environment variables
ENV JAVA_OPTS="-Djava.util.prefs.systemRoot=${USER_HOME}/.java -
Djava.util.prefs.userRoot=${USER_HOME}/.java/.userPrefs" \
   WORKING_DIRECTORY=${USER_HOME} \
   WSO2_SERVER_HOME=${WSO2_SERVER_HOME}

# expose ports
EXPOSE 4000 9763 9443

# initiate container and start WSO2 Carbon server
ENTRYPOINT ["/home/wso2carbon/init.sh"]
```

WSO2 identity server configuration

```
[server]
hostname = "localhost"
node_ip = "127.0.0.1"
base_path = "https://$ref{server.hostname}:${carbon.management.port}"

[super_admin]
username = "admin"
password = "admin"
create_admin_account = true

[user_store]
type = "database_unique_id"
properties.CaseInsensitiveUsername = false

[database.identity_db]
type = "postgre"
hostname = "postgres"
name = "identitydb"
username = "regadmin"
password = "regadmin"
port = "5432"

[database.shared_db]
type = "postgre"
hostname = "postgres"
name = "wso2shared"
username = "regadmin"
```

```
password = "regadmin"
port = "5432"


[keystore.primary]
file_name = "wso2carbon.jks"
password = "wso2carbon"

# custom identity server as key manager configurations
[[event_listener]]
id = "token_revocation"
type = "org.wso2.carbon.identity.core.handler.AbstractIdentityHandler"
name = "org.wso2.is.notification.ApimOauthEventInterceptor"
order = 1
[event_listener.properties]
notification_endpoint = "https://is:9443/internal/data/v1/notify"
username = "${admin.username}"
password = "${admin.password}"
'header.X-WSO2-KEY-MANAGER' = "WSO2IS"

[[resource.access_control]]
context = "(.)/keymanager-operations/user-info/claims(.)"
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]
context = "(.*)/keymanager-operations/user-info/claims/generate"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/applicationmgt/create"
scopes = "internal_application_mgt_create"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
```

```
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/applicationmgt/view"
scopes = "internal_application_mgt_view"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "PUT"
permissions = "/permission/admin/manage/identity/applicationmgt/update"
scopes = "internal_application_mgt_update"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "DELETE"
permissions = "/permission/admin/manage/identity/applicationmgt/delete"
scopes = "internal_application_mgt_delete"

[tenant_context.rewrite]
custom_webapps = ["/keymanager-operations/"]

[config_data]
path = "/_system/is_as_km/config"
```

WSO2 API manager

Dockerfile

```
FROM adoptopenjdk/openjdk11:jdk-11.0.8_10-alpine
# set Docker image build arguments
# build arguments for user/group configurations
ARG USER=wso2carbon
ARG USER_ID=802
ARG USER_GROUP=wso2
ARG USER_GROUP_ID=802
ARG USER_HOME=/home/${USER}
# build arguments for WSO2 product installation
ARG WSO2_SERVER_NAME=wso2am
ARG WSO2_SERVER_VERSION=3.2.0
ARG WSO2_SERVER_REPOSITORY=product-apim
ARG WSO2_SERVER=${WSO2_SERVER_NAME}-${WSO2_SERVER_VERSION}
ARG WSO2_SERVER_HOME=${USER_HOME}/${WSO2_SERVER}
```

```
ARG WSO2_SERVER_DIST_URL=https://github.com/wso2/${WSO2_SERVER_REPOSITORY}/releases/download/v${WSO2_SERVER_VERSION}/${WSO2_SERVER}.zip
ARG DNS_JAVA_VERSION=2.1.8
ARG K8S_MEMBERSHIP_SCHEME_VERSION=1.0.7
ENV ENV=${USER_HOME}"/.ashrc"

# create the non-root user and group and set MOTD login message
RUN \
    addgroup -S -g ${USER_GROUP_ID} ${USER_GROUP} \
    && adduser -S -u ${USER_ID} -h ${USER_HOME} -G ${USER_GROUP} ${USER} \
    && echo ${MOTD} > "${ENV}"

# create Java prefs dir
# this is to avoid warning logs printed by FileSystemPreferences class
RUN \
    mkdir -p ${USER_HOME}/.java/.systemPrefs \
    && mkdir -p ${USER_HOME}/.java/.userPrefs \
    && chmod -R 755 ${USER_HOME}/.java \
    && chown -R ${USER}:${USER_GROUP} ${USER_HOME}/.java

# install required packages
RUN \
    apk add --no-cache \
        bash \
        libxml2-utils \
        netcat-openbsd

# RUN mkdir -p ${WSO2_SERVER_HOME}
# COPY wso2am/ ${WSO2_SERVER_HOME}/
# RUN chown ${USER}:${USER_GROUP} -R ${WSO2_SERVER_HOME}
# RUN \
#     mkdir ${USER_HOME}/wso2-tmp \
#    && bash -c 'mkdir -p ${USER_HOME}/solr/{indexed-data,database}' \
#    && chown ${USER}:${USER_GROUP} -R ${USER_HOME}/solr \
#            && cp -r ${WSO2_SERVER_HOME}/repository/deployment/server/synapse-configs ${USER_HOME}/wso2-tmp \
#            && cp -r ${WSO2_SERVER_HOME}/repository/deployment/server/executionplans ${USER_HOME}/wso2-tmp
# add the WSO2 product distribution to user's home directory
RUN  wget --no-check-certificate -O ${WSO2_SERVER}.zip "https://github.com/wso2/product-apim/releases/download/v${WSO2_SERVER_VERSION}/${WSO2_SERVER}.zip"
RUN unzip -o -d ${USER_HOME} ${WSO2_SERVER}.zip \
```

```
   && chown ${USER}:${USER_GROUP} -R ${WSO2_SERVER_HOME} \
   && mkdir ${USER_HOME}/wso2-tmp \
   && bash -c 'mkdir -p ${USER_HOME}/solr/{indexed-data,database}' \
   && chown ${USER}:${USER_GROUP} -R ${USER_HOME}/solr \
   && cp -r ${WSO2_SERVER_HOME}/repository/deployment/server/synapse-configs ${USER_HOME}/wso2-tmp \
   && cp -r ${WSO2_SERVER_HOME}/repository/deployment/server/executionplans ${USER_HOME}/wso2-tmp \
   && rm -f ${WSO2_SERVER}.zip
# add libraries for Kubernetes membership scheme based clustering
ADD --chown=wso2carbon:wso2 https://repo1.maven.org/maven2/dnsjava/dnsjava/${DNS_JAVA_VERSION}/dnsjava-${DNS_JAVA_VERSION}.jar ${WSO2_SERVER_HOME}/repository/components/lib
ADD --chown=wso2carbon:wso2 http://maven.wso2.org/nexus/content/repositories/releases/org/wso2/carbon/kubernetes/artifacts/kubernetes-membership-scheme/${K8S_MEMBERSHIP_SCHEME_VERSION}/kubernetes-membership-scheme-${K8S_MEMBERSHIP_SCHEME_VERSION}.jar ${WSO2_SERVER_HOME}/repository/components/dropins

COPY --chown=wso2carbon:wso2 files/dropins ${WSO2_SERVER_HOME}/repository/components/dropins/
# set the user and work directory
COPY docker-entrypoint.sh ${USER_HOME}/
RUN chown ${USER}:${USER_GROUP} ${USER_HOME}/docker-entrypoint.sh
USER ${USER_ID}
WORKDIR ${USER_HOME}

COPY deployment.toml ${WSO2_SERVER_HOME}/repository/conf/
# RUN chown ${USER}:${USER_GROUP} ${WSO2_SERVER_HOME}/repository/conf/deployment.toml
# set environment variables
# set the user and work directory
USER ${USER_ID}
WORKDIR ${USER_HOME}

# set environment variables
ENV WORKING_DIRECTORY=${USER_HOME} \
   WSO2_SERVER_HOME=${WSO2_SERVER_HOME}

# expose ports
EXPOSE 9763 9443 9999 11111 8280 8243 5672 9711 9611 9099

# initiate container and start WSO2 Carbon server
ENTRYPOINT ["/home/wso2carbon/docker-entrypoint.sh"]
```

WSO2 API manager configuration

```
[server]
hostname = "localhost"
node_ip = "127.0.0.1"
#offset=0
mode = "single" #single or ha
base_path = "${carbon.protocol}://${carbon.host}:${carbon.management.port}"
#discard_empty_caches = false
server_role = "default"

[super_admin]
username = "admin"
password = "admin"
create_admin_account = true

[user_store]
type = "database_unique_id"

[database.apim_db]
type = "postgre"
hostname = "postgres"
name = "wso2apim"
username = "regadmin"
password = "regadmin"
port = "5432"

[database.shared_db]
type = "postgre"
hostname = "postgres"
name = "wso2shared"
username = "regadmin"
password = "regadmin"
port = "5432"

[keystore.tls]
file_name =  "wso2carbon.jks"
type =  "JKS"
password =  "wso2carbon"
alias =  "wso2carbon"
key_password =  "wso2carbon"

[keystore.primary]
```

```
file_name =  "wso2carbon.jks"
type =  "JKS"
password =  "wso2carbon"
alias =  "wso2carbon"
key_password =  "wso2carbon"

[keystore.internal]
file_name =  "wso2carbon.jks"
type =  "JKS"
password =  "wso2carbon"
alias =  "wso2carbon"
key_password =  "wso2carbon"

[[apim.gateway.environment]]
name = "Production and Sandbox"
type = "hybrid"
display_in_api_console = true
description = "This is a hybrid gateway that handles both production and sandbox token traffic."
show_as_token_endpoint_url = true
service_url = "https://localhost:${mgt.transport.https.port}/services/"
username= "${admin.username}"
password= "${admin.password}"
ws_endpoint = "ws://localhost:9099"
wss_endpoint = "wss://localhost:8099"
http_endpoint = "http://localhost:${http.nio.port}"
https_endpoint = "https://localhost:${https.nio.port}"

[apim.analytics]
enable = false
#store_api_url = "https://am-analytics-worker:7444"
#username = "$ref{super_admin.username}"
#password = "$ref{super_admin.password}"
#event_publisher_type = "default"
#event_publisher_impl                                                    =
"org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataBridgeDataPublisher"
#publish_response_size = true

# [[apim.analytics.url_group]]
# analytics_url =["tcp://am-analytics-worker:7612"]
# analytics_auth_url =["ssl://am-analytics-worker:7712"]
#type = "loadbalance"

#[[apim.analytics.url_group]]
```

```
#analytics_url =["tcp://analytics1:7612","tcp://analytics2:7612"]
#analytics_auth_url =["ssl://analytics1:7712","ssl://analytics2:7712"]
#type = "failover"

[apim.key_manager]
service_url = "https://is:${mgt.transport.https.port}/services/"
type = "WSO2-IS"

[apim.jwt]
enable = true
encoding = "base64" # base64,base64url
#generator_impl = "org.wso2.carbon.apimgt.keymgt.token.JWTGenerator"
claim_dialect = "http://wso2.org/claims"
#convert_dialect = false
#header = "X-JWT-Assertion"
signing_algorithm = "SHA256withRSA"
#enable_user_claims = true
#claims_extractor_impl = "org.wso2.carbon.apimgt.impl.token.ExtendedDefaultClaimsRetriever"


[apim.devportal]
url = "https://localhost:${mgt.transport.https.port}/devportal"
#enable_application_sharing = false
#if application_sharing_type, application_sharing_impl both defined priority goes to application_sharing_impl
#application_sharing_type = "default" #changed type, saml, default #todo: check the new config for rest api
#application_sharing_impl = "org.wso2.carbon.apimgt.impl.SAMLGroupIDExtractorImpl"
#display_multiple_versions = false
#display_deprecated_apis = false
#enable_comments = true
#enable_ratings = true
#enable_forum = true
#enable_anonymous_mode=true

[apim.cors]
allow_origins = "*"
allow_methods = ["GET","PUT","POST","DELETE","PATCH","OPTIONS"]
allow_headers = ["authorization","Access-Control-Allow-Origin","Content-Type","SOAPAction","apikey", "testKey"]
allow_credentials = false

[[event_handler]]
```

```
name="userPostSelfRegistration"
subscriptions=["POST_ADD_USER"]

[service_provider]
sp_name_regex = "^[\\sa-zA-Z0-9._-]*$"

[database.local]
url = "jdbc:h2:./repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE"

[[event_listener]]
id = "token_revocation"
type = "org.wso2.carbon.identity.core.handler.AbstractIdentityHandler"
name = "org.wso2.is.notification.ApimOauthEventInterceptor"
order = 1
[event_listener.properties]
notification_endpoint = "https://localhost:${mgt.transport.https.port}/internal/data/v1/notify"
username = "${admin.username}"
password = "${admin.password}"
'header.X-WSO2-KEY-MANAGER' = "default"

[config_data]
path = "/_system/apim/config"

[transport.passthru_https.sender.parameters]
HostnameVerifier="AllowAll"
```