ESSENTIAL DOCKER INSTALL GUIDE

Version: 2.9

Date: 25th July 2023

Contents

INTRODUCTION

This user describes the process for deploying an instance of the Essential Docker platform in a Docker container. This Docker version provides a stand-alone deployment of the Essential Docker platform with a single, pre-provisioned tenant.

PRE-REQUISITES

Before starting them deployment of the Essential Docker solution, please ensure that you have the following elements in place:

- A Linux runtime platform **that is supported by Docker**. For more information on those supported platforms, please see https://www.docker.com/get-docker#/edition
- An installation of Docker for your chosen Linux platform
- A valid SSL/TLS certificate for the hostname of the server
- A valid license key for Essential Docker

IMPORTANT: ESSENTIAL DOCKER WILL NOT FUNCTION WITHOUT A VALID SSL/TLS CERTIFICATE. DO NOT PROCEED WITH THE INSTALLATION WITHOUT A VALID CERTIFICATE.

This runtime platform can be hosted within a cloud service, such as Amazon AWS. Note that Windows Docker host platforms are not supported, currently.

IMPORTANT NOTE ABOUT RHEL.

Currently, Docker is not supported on RHEL and instead uses Podman for container support. We do not support the use of Essential Docker on RHEL using Podman as Podman has some resource limitations which prevent Essential software running normally.

Server Requirements

4 Core x86 CPU
64GB RAM
100GB Disk Space (SSD Recommended) **IMPORTANT!** Minimum 30GB for the partition on the Docker host server which stores the directory **/var/lib/docker** (typically the boot partition).

The following ports must be configured for access to the server

Inbound

Port 80 TCP (optional but recommended – will redirect to HTTPS 443)

Port 443 TCP

You will probably also open restricted access to port 22 for SSH to the server.

Outbound

The port used for SMTP email (465, 587, etc) in the configuration below

PLATFORM SECURITY AND HARDENING

Essential Docker is designed as an on-premise/self-hosted alternative to our Essential Cloud product and as such the **security and hardening of the host machine is your responsibility**. We strongly recommend the use of a **suitable web application firewall (WAF)** in front of Essential Docker to provide an additional security layer. All internal security testing of Essential Docker has been assessed with the assumption of a suitable WAF.

INSTALLATION COMPONENTS

Essential Docker is comprised of two components:

- **eipdata folder**, which ensures that all customer-specific content, including all the repositories and custom Views are maintained across multiple versions and upgrades of the Essential Docker software. This is delivered as a compressed Linux TAR file.
- **Essential Docker image**, that provides all of the software of the platform such that upgrades to this software can be easily made by deploying a new version of the Docker image.

INSTALLATION OVERVIEW

With your chosen Linux environment in place and Docker installed, the following steps describe how to install and configure the Essential Docker environment

1. Unpack the eipdata.zip file
2. Configure license and certificate
3. Deploy the Docker image
4. Run the preparation scripts
5. Accessing Essential Docker
6. Configure Platform Settings
7. Configure user authentication method (Local or SAML)

UNPACK EIPDATA.ZIP FILE

Upload the EIPDATA zip file to your home directory and unpack the file to the root of your Linux environment.

Example:

```
sudo cp ~/eipdata-6.11.zip /
```

> *NOTE: If you do not wish to store the EIPDATA folder at the root location, you should use a symbolic link from the root location (`/eipdata`) to your preferred location e.g. a different disk*

Unpack the eipdata.zip file into the root of your Linux environment.

Example:

```
cd /
```

```
sudo unzip eipdata-6.11.zip
```

Confirm that you have a valid directory structure in **/eipdata** as follows:

- /eipdata/
  - certificates
  - data
  - import
  - license-key
  - repositories
  - viewer

Example:

```
cd /eipdata
```

```
ls -la
```

CONFIGURE LICENSE AND CERTIFICATE

ESSENTIAL DOCKER LICENSE KEY

Before you can use Essential Docker, you must place the license key file that you received from EAS into the **/eipdata/license-key** folder, replacing the existing **license.key** file.

Example:

```
sudo cp ~/license.key /eipdata/license-key/
```

SSL CERTIFICATES

Regardless of where the host Linux platform is deployed, a HTTPS connection is required for all interactions with the Essential Docker service. These certificates must be valid for your deployment domain and signed by a certificate authority to ensure that web browsers will connect to the service.

Once you have acquired your signed certificate,

- Copy the certificate file to **/eipdata/certificates/ssl-certificate.crt**
- Copy the key file to **/eipdata/certificates/ssl-certificate.key**

Example:

```
sudo cp ~/ssl-certificate.crt /eipdata/certificates/ssl-certificate.crt
sudo cp ~/ssl-certificate.key /eipdata/certificates/ssl-certificate.key
```

> *NOTE: It is important that the certificate and key names are as shown above i.e.* ***ssl-certificate.crt*** *and* ***ssl-certificate.key***

ESSENTIAL DOCKER SCRIPTS

The Essential Docker scripts provide a simple set of controls for your environment. These can be deployed to any location on your host server must be run by a user with permission to run Docker commands:

- **initialise-api.sh** – Configures the API platform and produces an API Key
- **initialise-nosql.sh** - Prepare NoSQL DB for use with APIs
- **install-essential-docker-xxx.sh** - unpacks and loads the Essential Docker software
- **prepare-eipdata-for-api.sh** - Prepare tenant's permanent storage for API Platform and NoSQL DB
- **remove-essential-docker.sh** - stops and removes the Essential Docker software, e.g. in preparation for an upgrade
- **restart-docker-api.sh** – restarts the Essential Docker API
- **restart-docker-import.sh** - restarts the Essential Import Utility
- **restart-docker-viewer.sh** - restarts the Essential Viewers
- **restart-essential-docker.sh** - stops and then restarts the Essential Docker software
- **start-essential-docker.sh** - starts the Essential Docker environment
- **stop-essential-docker.sh** - stops the Essential Docker environment. *Note: this does not remove the installation*

UNPACKING THE DOCKER SCRIPTS

With your local settings complete, your platform is ready to load and run the Essential Docker system.

Using the Essential Docker scripts, deploy and start Essential Docker as follows.

- Log in to your Essential Docker Linux environment, using a terminal session
- In the root of your Linux environment, create a new directory called **essential.** *Example:*
  - `cd /`
  - `sudo mkdir essential`
- Copy and unpack the essential-docker-scripts.zip file into your new directory. *Example:*
  - `cd /essential`
  - `sudo cp ~/essential-docker-scripts.zip .`
  - `sudo unzip essential-docker-scripts.zip`

DEPLOY THE DOCKER IMAGE

- Upload the Docker image file for Essential Docker to your home directory in the Linux environment then copy or move this to the essential directory you created in the previous step. *Example*:
  - `sudo cp ~/essentialdocker.vXYZ.tar.gz /essential`

STARTING ESSENTIAL DOCKER

- Switch to your essential install directory
  - **`cd essential`**
- In this directory, install the Essential Docker system, using the command:
  - **`sudo ./install-essential-docker.sh`**
- Once the install process has completed, start the Essential Docker, using the command:
  - **`sudo ./start-essential-docker.sh`**

> *NOTE: The application start-up process **takes between 3 and 5 minutes to complete** before the system is ready to use. Attempts to access the application during this time may result in an error page being displayed.*

To display the running Docker container(s), use:

- **`sudo docker ps`**

CREATE API KEY AND INITIALISE API GATEWAY

Once the **/eipdata** directory has been prepared, you are ready to start your Essential Docker installation.

> *Note: Essential Docker must have started before the following steps can be completed.*

Once Essential Docker has started and you have logged in, you can create a new API key that must be used in every request to the Essential Docker API, ensuring that only authorized requests will be accepted.

Run the 'initialise-api.sh' script.

- **`cd /essential`**
- **`sudo ./initialise-api.sh`**

This will return details of your newly created API key. You should take a note of both:

- keyId
- keySecret

and combine these in the following pattern (*note the colon*) in each request, as your API key:

- ***keyId:keySecret***

The API key must be specified in every request to the API using the HTTP request header:

- **x-api-key**

ACCESSING ESSENTIAL DOCKER

It is assumed that Essential Docker is deployed to an empty Linux server, in particular that there is no other HTTP web server running on that server outside of the Essential Docker.

Access the Essential Docker application using the following URL pattern:

https://yourhostname

For example, if the Essential Docker platform is deployed on a server with hostname of **essential.example.com**, access Essential Docker, using:

https://essential.example.com

> *NOTE: It is important that the server has a valid SSL/TLS certificate for the URL on which Essential Docker is accessed. A missing or invalid certificate will result in Essential being unable to run correctly.*

CONFIGURE PLATFORM SETTINGS

PLATFORM HOSTNAME

As part of the password workflows in Essential Docker, the platform must be able to send emails with links to the application. For Essential Docker, the fully-qualified hostname of the server on which the platform is running must be set in the application.

Log into the Essential Docker application using the pre-configured account

username: first.run@enterprise-architecture.org

password: Admin1

1. Select the **System Administration** option from the **Configure** Menu.
2. Select the **Platform Settings** menu to display the following form, as shown in Figure 1 Platform Settings to specify the domain name of the Docker deployment.

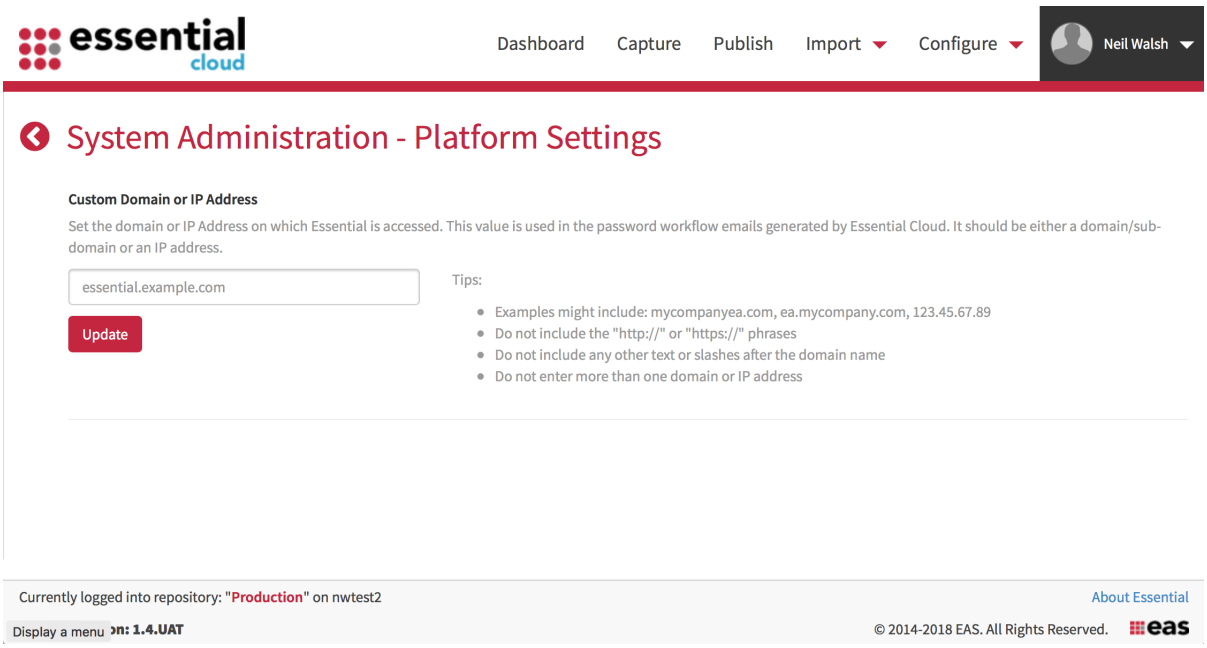


Figure 1 Platform Settings to specify the domain name of the Docker deployment

Specify the domain name, e.g. *essential.example.com*, of the server on which your Essential Docker installation is deployed and click the "**Update**" button.

SETTING UP THE DEFAULT VALUES FOR NEW USERS

1. Select the **System Administration** option from the **Configure** Menu
2. Select **Default Values for New Users**
3. Check all boxes on this screen

> *Once you have created your initial administrator users, you should amend these values to any defaults you would like to use when provisioning regular (non-admin) users*

SET THE API KEY IN API CONFIGURATION

With the API key details created earlier, you can now use these to complete the API Configuration of Essential Docker. This enables Essential Viewer components to function correctly and allows you to use the API without having to hard-code an API key into your custom View code. You must complete this step even if you do not intend to use the API directly

Log into Essential Docker using an administrator account and select the **Configure** -> **System Administration** option from the menu bar. In **System Administration**, select **Platform Settings**. Paste your full API key – in the format ***keyId:keySecret*** – into the API Configuration panel and push the '**Update'** button.

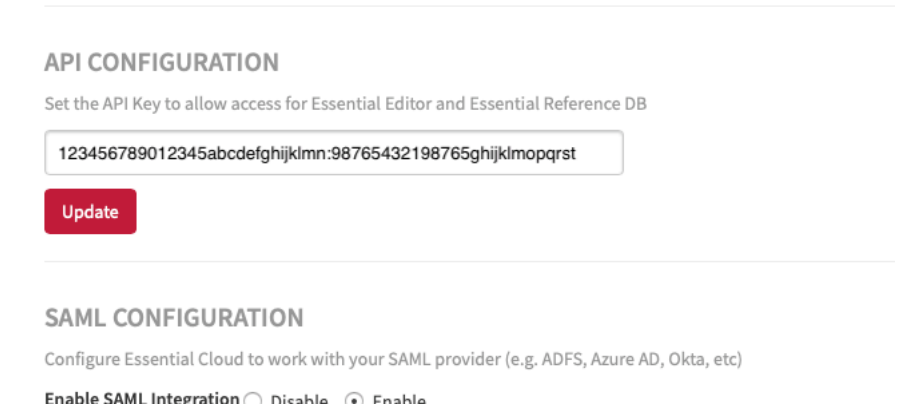


Figure 2 Set the API key in API Configuration

CONFIGURE THE ESSENTIAL IMPORT UTILITY

From Essential Docker 2.2 onwards, the Essential Import Utility must be configured to specify which repository to use as a reference for the meta model. This step is required even if you are upgrading from earlier versions of Essential Docker.

Rather than loading an exported copy of current repository, the Essential Import Utility now connects directly to the Essential Docker server to receive details of the Essential Meta Model, including any custom classes and slots that may be available. To specify which of the repositories in the Essential Docker server should be used by the Import Utility as the reference repository:

- Log into Essential Docker and select **Import** -> **Import Utility**
- Select the **Import Settings** tab
- In the **Reference Repositories** section, check the tick-box to select the Essential Docker repository that contains the meta model items that you require e.g. Production

> *If you cannot select any classes or slots in the Import Specification editor, make sure that the Reference Repositories section has the correct reference repository selected.*

CONFIGURING THE USER AUTHENTICATION METHOD

There are three options for managing user identities in Essential Docker.

1. Local accounts using the built in Identity Provider
2. Federated identity using SAML to integrate with your organisation's directory
3. Combination of both local and SAML

Any use of **Local accounts** requires the configuration of an SMTP email server as detailed below.

Any use of **SAML accounts** requires the configuration of SAML as detailed below.

CONFIGURING THE EMAIL SERVER FOR LOCAL ACCOUNTS

In support of user provisioning and self-service password reset, the Essential Docker environment will send emails. These emails contain a link with a token that is used to reset their password. The Essential Docker environment uses your own email (SMTP) server to avoid any issues with spam or receiving email from unknown network addresses.

Stop Essential Docker

- **`cd /essential`**
- **`sudo ./stop-essential-docker.sh`**

Set the email configuration by editing the file, **output-event-adapters.xml**, which is found in **`/eipdata/data/wso2data`**

Example:

```
sudo nano /eipdata/data/wso2data/output-event-adapters.xml
```

Open this file and look for the tag:

```
<adapterConfig type="email">
```

Within this block, the following tags must be updated:

```
<property key="mail.smtp.from">
    ENTER THE FRIENDLY NAME FOR THE EMAIL ACCOUNT
</property>
<property key="mail.smtp.user">
    ENTER THE SMTP USER NAME FOR THE ESSENTIAL DOCKER
</property>
<property key="mail.smtp.password">
    ENTER THE SMTP USER PASSWORD FOR ESSENTIAL DOCKER
</property>
<property key="mail.smtp.host">
    ENTER HOSTNAME OF YOUR SMTP SERVER, e.g. smtp.myserver.com
</property>
<property key="mail.smtp.port">
    ENTER THE PORT NUMBER ON WHICH YOUR SMTP SERVER IS LISTENING, e.g. 587
</property>
```

Within each tag, replace the "ENTER THE …" statements with your email settings, e.g.

- **mail.smtp.from** – the friendly name for the email account for the Essential Docker platform e.g. essentialcloud@example.com, Essential Docker or The EA Team
- **mail.smtp.user** – the user name which the Essential Docker platform uses to authenticate to your email server e.g. essentialcloud@example.com
- **mail.smtp.password** – the password for the Essential Docker email account
- **mail.smtp.host** – the hostname of your SMTP server e.g. smtp.example.com
- **mail.smtp.port** – the network port on which to connect to your SMTP server, e.g. 587

Once you have entered your settings, save the file back to **/eipdata/data/wso2data/ output-event-adapters.xml**.

Example:

In the Nano editor, Ctrl+O saves the file

> ***NOTE:*** *Make sure that there are no trailing spaces in the values that you set for these email server properties, e.g. the mail.smtp.host setting*

```
<property key="mail.smtp.host">mail.myserver.com</property>
```

Start Essential Docker

- `cd /essential`
- `sudo ./start-essential-docker.sh`

CREATING THE FIRST USER

1. Login to Essential using the “first run” user
    - **username:** first.run@enterprise-architecture.org
    - **password:** Admin1
2. Select the **System Administration** option from the **Configure** Menu
3. Select **Manage Users**
4. Add a new user with System Admin permissions
5. Logout of the application
6. From the login screen, click **First Time User** and follow the directions to set your password
7. Once you have successfully logged in under you own account, you should delete the First Run user

> *We strongly recommend deleting the first run user once you have a working System Admin user using you own details. The First Run user uses a default password shared that is highly insecure.*

CONFIGURING ESSENTIAL DOCKER FOR SAML

See **Appendix 3** in this document for details on setting up your SAML Provider to work with Essential Docker

STARTING AND STOPPING ESSENTIAL DOCKER

In case of a system restart for the platform – e.g. your Linux system – on which you are hosting Essential Docker, you will need to restart the Essential Docker system. To start the system, use:

- **sudo ./start-essential-docker.sh**

In advance of a scheduled platform shutdown, the Essential Docker system can be stopped using the command:

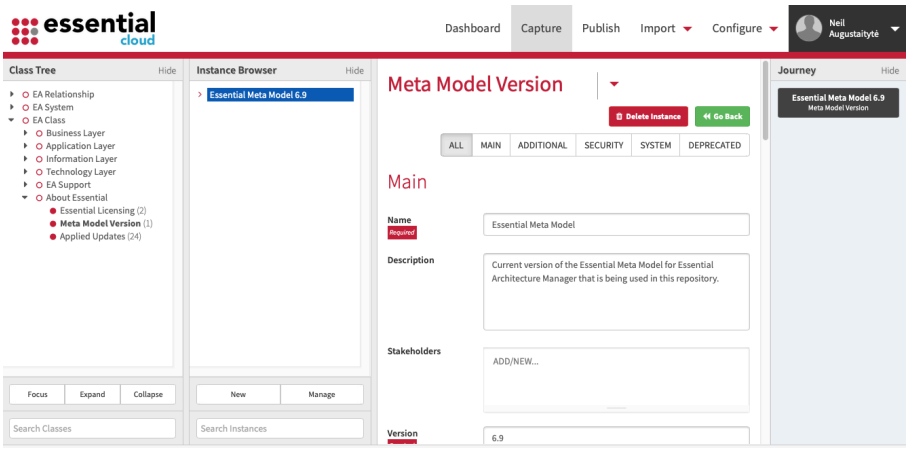
- **sudo ./stop-essential-docker.sh**

UPDATING THE ESSENTIAL META-MODEL

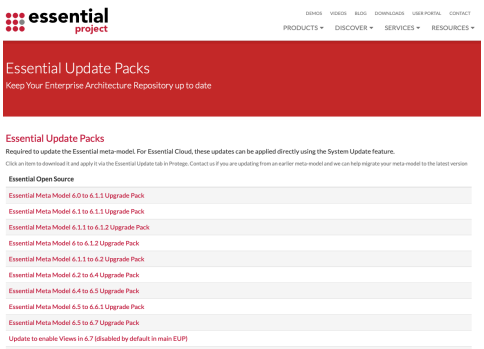
Over time, newer versions of the Essential Meta-Model are released. Essential Docker comes with a default meta-model however this may not necessarily be the latest version.

You should update the meta-model using the Essential Update Packs (EUPs) available on the Essential Project website at https://www.enterprise-architecture.org/update_packs.php

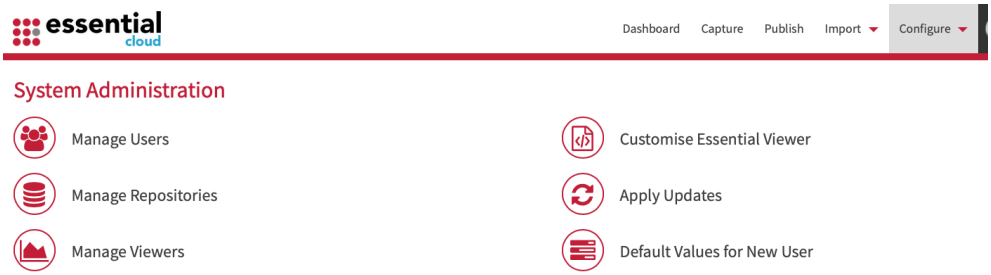
1. First check the version of the meta-model you currently have:
    a. Click Capture
    b. Browse to **EA Class** > **About Essential** > **Meta-Model Version**
    c. Make a note of the version



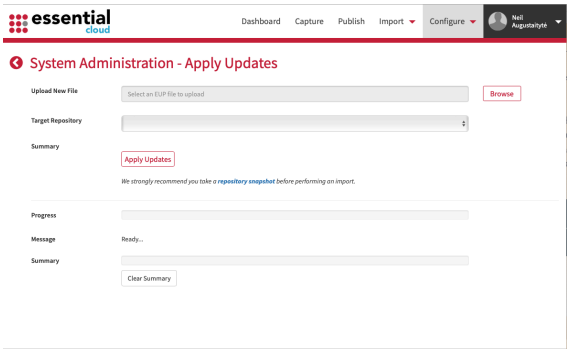
2. Visit the Essential Project website and download all the relevant updates to take you from your current version to the latest version. The update must be applied in the correct order. https://www.enterprise-architecture.org/update_packs.php



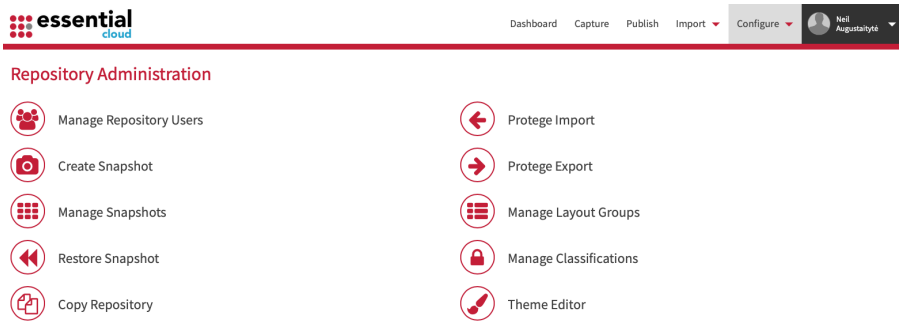
3. Next, browse to **Configure** > **System Administration**



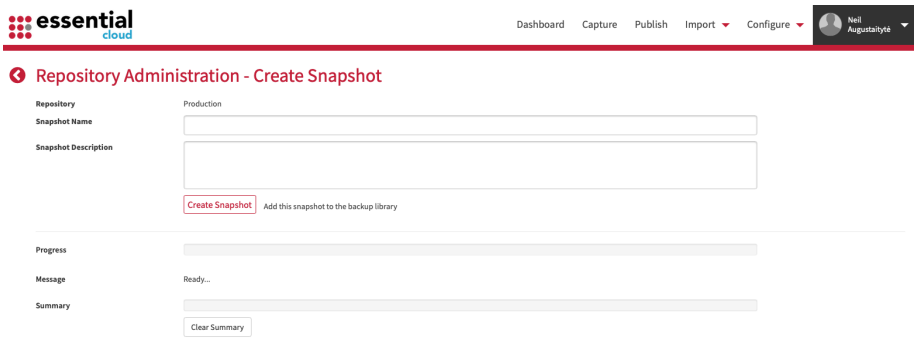
4. Click on **Apply Updates**



5. Click the Browse button to select the update you wish to apply. If this is your initial setup process, then select **Production** as the target repository.
6. Click Apply Updates and wait for the update to complete.
7. Repeat the select and apply process for each EUP.
8. Once all the updates have been applied, check you have the **Production** repository selected then click **Configure** > **Repository Administration**



9. Next select Create Snapshot and give the snapshot a meaningful name and description. For example:
    a. Name: Essential Baseline v6.8
    b. Description: An empty baseline repository with the meta-model at version 6.8



10. The Essential Meta-Model should now be at the latest version. You should repeat this process for each repository you wish to update or use the **Repository Admin** > **Restore Snapshot** for each repository.

You can use the above process to update the meta-model at any time and we recommend taking snapshots before and after the update process.

UPGRADING TO A NEWER VERSION OF ESSENTIAL DOCKER

The Docker image file that is delivered as part of the Essential Docker solution provides a complete, configured installation of Essential Docker. Keeping a copy of each released version of this Docker image provides a backup of the previous version to which you can revert in case of any issues with a new release.

All user data, including repository content, certificates, license key, user accounts and custom views are managed ***outside*** of the Docker image and are unaffected by the deployment of new versions.

> *Backups of the `/eipdata` folder are beyond the scope of the Essential Docker solution. We recommend that you arrange scheduled backups of the `/eipdata` folder on your platform.*

Consult the Essential Docker Upgrade Guide available from the Essential Project website for detailed instructions.

https://www.enterprise-architecture.org/docker_download.php

TROUBLESHOOTING

ESSENTIAL DOCKER IMAGES

Should the Essential Docker applications not respond as expected, you can confirm that Docker has loaded this successfully, using the Docker image command:

```
sudo docker images
```

To display the running Docker container(s), use:

```
sudo docker ps
```

ACCESSING ESSENTIAL DOCKER LOGS

Essential Docker is a fully-supported platform and if you experience any issues with the solution, your first point of contact should be support@enterprise-architecture.org.

Occasionally, you may wish to access the log files of the components of the solution.

You can see startup logs for Docker by using the command

- `sudo docker logs essentialcloud`

You can log into the Docker container to see specific logs using the command:

- `sudo docker exec -it essentialcloud bash`

From this command-line environment, you can find the log files for the components of the solutions, as follows:

- Web proxy server logs (NGINX):
  - `/var/log/nginx/`
- Essential Data Manager application:
  - `/opt/tomcat-edm/logs`
- Essential Import Utility application:
  - `/opt/tomcat-importutility/logs`
- Essential Viewer application:
  - `/opt/tomcat-viewer/logs`
- Essential Login Service application:
  - `/opt/tomcat-edm/logs`
- Identify Provider Logs:
  - `/opt/wso2/repository/logs`

GETTING ADDITIONAL HELP

For deployment issues contact support@enterprise-architecture.org
For questions regarding using Essential (e.g. modelling or Views) visit https://www.enterprise-architecture.org/support

APPENDIX 1 - ESSENTIAL API PLATFORM – OAUTH 2.0 SECURITY MODEL

The Essential API Platform uses the OAuth 2.0 bearer token workflow, in addition to an API Key, to authenticate and authorise every request to the Essential API.

Before making a request to the API, you must get a valid bearer token and include this bearer token in every request to the API. A bearer token remains valid for a limited time – five minutes – after which any request that uses this token will be rejected. When that request is rejected, with an HTTP 403 response containing error code 10, a new bearer token must be requested. Figure 3 API Integration Workflow summarises the integration workflow.

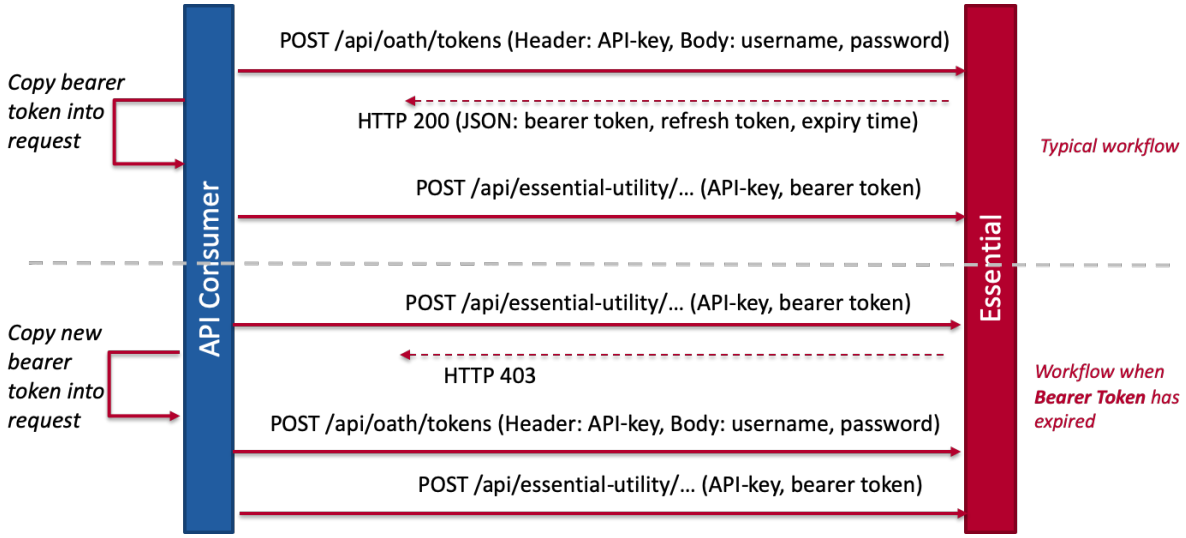


Figure 3 API Integration Workflow

Bearer tokens are requested from the OAuth/Token API, which is accessed at: **Error! Hyperlink reference not valid.**

The username and password should be the username and password of a valid user account within the Essential Docker environment, with System Administration rights. This could be an existing user account or an account defined for external systems to for automated integration.

EXAMPLE REQUESTS AND RESPONSES

Request a bearer token:

```
curl -X POST \
  https://myhostname/api/oauth/token \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: 123456789:123456789' \
  -d '{
    "grantType":"password",
    "username":"****",
```

```
    "password": "******"
}'
```

Returns 200 OK and bearer token details in the response:

```
{
    "bearerToken": "9AeX94795cIz5rE1A30%2FSt2g%3D%3D",
    "refreshToken": "4eRp7%2BDhCoRR8QPg%3D%3D",
    "expiresInMinutes": 5
}
```

If invalid credentials are supplied, the OAuth Token API returns a status code of 500:

```
{
    "statusCode": 500,
    "message": {
        "errorCode": 0,
        "errorMessage": "Invalid credentials"
    }
}
```

Once a valid bearer token has been received from the OAuth API, include this in your API requests in the HTTP Request Headers:

- **`x-api-key`** – set to your API key
- **`Authorization`** – set to the format 'Bearer <THE BEARER TOKEN RECEIVED>', e.g.
    - `Authorization: Bearer 9AeX94795cIz5rE1A30%2FSt2g%3D%3D`

Example request to the Essential Utility API

```
curl -X POST \
  https://myhostname/api/essential-
utility/v2/repositories/abc12345/classes/Data_Object/instances \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: 123456789:123456789' -H 'Authorization: Bearer
9AeX94795cIz5rE1A30%2FSt2g%3D%3D' \
  -d '{
    "instances": [{data-object details}, {data-object details}]
}'
```

returns HTTP 200 OK and the response body.

If the bearer token has expired, the API responds with HTTP 403 Forbidden and response body:

```
{
    "statusCode": 403,
    "message": {
```

```
        "errorCode": 10,
        "errorMessage": "Invalid token: the bearer token used in this
request has expired."
    }
}
```

If HTTP 403 Forbidden is received, make a request for a new bearer token and retry the API request, using the new token.

APPENDIX 2 - ESSENTIAL NOSQL DATABASE

CONCEPT

The Essential NoSQL Database is included in the Essential Docker installation, providing a platform in which additional related content can be captured and managed alongside the Essential repository.

Content in the NoSQL database is stored and managed as JSON documents and can be related to each other and also to elements that exist within the Essential repository. The content that can be stored and the structure of this content is user defined. Users can define as many collections as are required, with flexibility about the structure of the data models used – either embedded documents (denormalized) model or a related documents (normalized) model or combinations of both.

The documents in the NoSQL database can be used in custom Essential Views that understand the content and structure of the relevant NoSQL collections and combine this with content from the Essential repository.

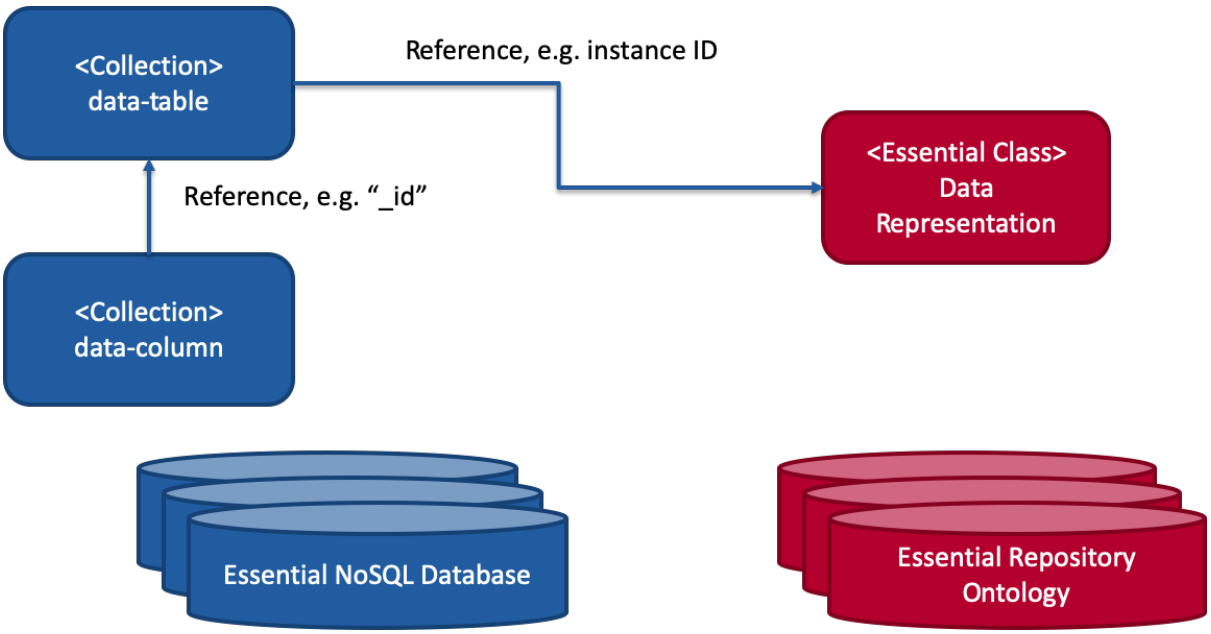


Figure 4 Relating NoSQL DB elements to each other and the Essential Ontology

DATABASES AND COLLECTIONS

Databases in the Essential NoSQL Database manages a group of collections and each collection contains a set of JSON documents, e.g. a database might contain a collection for Data Tables (data-tables) and a collection for Data Columns (data-columns). New collections can be defined dynamically via the API by making a request to that collection. Multiple databases can be defined but need not necessarily be aligned 1-to-1 to the Essential repositories.

DATABASE NAMING

The name of the NoSQL database forms part of the URI within the Essential Reference API, so any name that is used must conform to URI standards.

COLLECTION NAMING

Collections are defined dynamically, using names that conform to URI standards. To create a new collection, simply make a POST request to the collection and it will be created automatically.

Each collection contains zero or more JSON documents. These documents can contain any valid JSON content.

ADD THE API KEY TO THE ESSENTIAL DOCKER PLATFORM PROPERTIES

To ensure that the Essential Viewer can access the APIs, make sure that your API key has been captured in the Essential Docker Platform Properties. This is managed from within Essential Docker, System Properties / Platform Properties. See Set the API Key in API Configuration.

ESSENTIAL REFERENCE API

The essential reference API provides access to read to and write to the Essential NoSQL database. The Essential Reference API is split into two parts:

- **Essential Reference Batch** – which provides an API to add and remove collections from a NoSQL database
- **Essential Reference** – which provides an API to query to the NoSQL database

ESSENTIAL REFERENCE BATCH

The Reference Batch API adds and removes collections to and from a NoSQL database. This API is designed to operate with very large volumes of JSON data. Adding millions of documents may take a few minutes, depending on the size of each document and the existing content of the target database.

In the API URLs, the following resource types are defined:

- **stores** – which represent the databases in the NoSQL environment
- **collections** – which represent the collections in the NoSQL store (database)

Example: Add the included JSON elements to the data-object-attributes collection in the 'tenant-1' store:

POST

```
POST https://<your-essential-cloud-host>/api/essential-reference-batch/v1/stores/tenant-1/collections/data-objects
```

Elements in bold are variables, that here define the NoSQL Database and Collection into which new elements are to be loaded.

The JSON payload can include any JSON documents. However, in the POST request, the documents to POST **must** be wrapped in a JSON document, as follows:

```
{"instances": [ {element JSON document}, {element JSON document}…]}
```

For example, the following document creates new Data Objects in the NoSQL database:

```
{"instances": [
    {
        "externalIds": [{"sourceName": "Test Data Table 001"}],
        "className": "Data_Object",
        "name": "Data_Table_001",
        "description": "Description of Data Table 001",
        "slots": [{
            "slotName": "contained_data_attributes",
            "slotValue": ["data_object_attribute_1",
"data_object_attribute_2", "data_object_attribute_3",
"data_object_attribute_4"]
        }]
    },
    {
        "externalIds": [{"sourceName": "Test Data Table 002"}],
        "className": "Data_Object",
```

```
        "name": "Data_Table_002",
        "description": "Description of Data Table 002",
        "slots": [{
            "slotName": "contained_data_attributes",
            "slotValue": ["data_object_attribute_5",
"data_object_attribute_6", "data_object_attribute_7",
"data_object_attribute_8"]
        }]
    },
    {
        "externalIds": [{"sourceName": "Test Data Table 003"}],
        "className": "Data_Object",
        "name": "Data_Table_003",
        "description": "Description of Data Table 003",
        "slots": [{
            "slotName": "contained_data_attributes",
            "slotValue": ["data_object_attribute_9",
"data_object_attribute_10", "data_object_attribute_11",
"data_object_attribute_12"]
        }]
    },
    {
        "externalIds": [{"sourceName": "Test Data Table 004"}],
        "className": "Data_Object",
        "name": "Data_Table_004",
        "description": "Description of Data Table 004",
        "slots": [{
            "slotName": "contained_data_attributes",
            "slotValue": ["data_object_attribute_13",
"data_object_attribute_14", "data_object_attribute_15",
"data_object_attribute_16"]
        }]
    },
    {
        "externalIds": [{"sourceName": "Test Data Table 005"}],
        "className": "Data_Object",
        "name": "Data_Table_005",
        "description": "Description of Data Table 005",
        "slots": [{
            "slotName": "contained_data_attributes",
            "slotValue": ["data_object_attribute_17",
"data_object_attribute_18", "data_object_attribute_19",
"data_object_attribute_20"]
        }]
    }
]}
```

On success, an HTTP / 201 response is returned with a JSON message in the body that reports the updated size of the collection. E.g.:

```
201 Created
```

```
{"message": "Inserted new instances to collection: data-object-attributes.
Now contains 5 instances"}
```

If an error is encountered, an error HTTP response code with an error message will be returned.

- **400 – bad request**, indicating that there was a problem with the request that was received by the server, e.g. no JSON payload was supplied in the 'json-file' request body
- **500 – internal server error**, indicating that the server encountered a problem whilst servicing the request. Check your configuration and that all the elements of Essential Docker are running correctly

DELETE

Delete operations are available to remove either a specific document from a collection or a whole collection.

```
DELETE https://<your-essential-cloud-host>/api/essential-reference-batch/v1/stores/tenant-1/collections/data-objects
```

Elements in bold are variables, that here define the NoSQL Database and Collection from which the documents are to be deleted.

On success, an HTTP 200 / OK response is returned with a JSON body that provides further information about the delete operation. E.g.

```
{"message": "Removed 25 instances of data-object-attributes"}
```

In addition to deleting all the documents in a collection, specific documents can be deleted from the collection, identified by the ID of that document. The 'instances' element of the URL is used to specify which document is to be deleted from the specified collection.

```
DELETE https://<your-essential-cloud-host>/api/essential-reference-batch/v1/stores/tenant-1/collections/data-objects/instances/5ceec7506f09fc411777e87c
```

Elements in bold are variables, that here define the NoSQL Database and Collection from which the documents are to be deleted.

On success, an HTTP 200 / OK response is returned with a JSON body that confirms the object that was removed from the requested collection. E.g.

```
{"message": "Removed object: 5ceec7506f09fc411777e87c from data-object-attributes"}
```

ESSENTIAL REFERENCE

The Essential Reference API provides a small suite of requests for querying the Essential NoSQL database, which can contain JSON documents of any metadata type. When relating documents – e.g. to connect a set of database table definitions to an Essential Data Object – you can use Essential IDs or NoSQL DB IDs interchangeably to identify the related elements. When relating to elements within NoSQL collections, use the

NoSQL database document ID and when relating to Essential repository elements, use the Essential IDs. These identifiers do not have to be synchronised between collections or within the Essential repository.

Requests to read (**GET**) documents, and create (**POST**) a **single** document, update (**PUT**) a **single** document in the selected collection are provided.

Four types of GET request are provided:

- GET all the instances (documents) in the specified collection
- GET a specific instance (document) in the specified collection
- GET the instances (documents) that are related to the specified instance (in an Essential repository or Essential NoSQL database) via a specific attribute of the JSON
- GET the instances (documents) that are related to the specified instance (in an Essential repository or Essential NoSQL database) via an Essential slots attribute (from the Essential Utility API)

All of the GET requests for a collection of instances return a JSON document of the following form:

```
{
    "instances": [{JSON document a}, {JSON document b}…]
}
```

If no instances exist in the specified collection, an empty list is returned, as follows:

```
{
    "instances": []
}
```

The request for a specific instance returns a single JSON document. The structure of this document is user-defined with an additional "_id" attribute that is assigned by the NoSQL database.

GET ALL INSTANCES IN THE SPECIFIED COLLECTION

Use the following format to make a GET request for all instances (documents) in a specific collection:

```
https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}/
```

- {store-name} is the name of a NoSQL database instance. This should match the store-name used when POST-ing JSON content into the NoSQL database
- {collection-name} specifies which collection is to be queried, matching a collection that has been populated by a POST request

Example:

```
https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-1/collections/data-object/
```

returns HTTP 200 / OK with the following JSON body:


```
{
    "instances": [
        {
            "_id": {
                "$oid": "5d0aaae2b225c42a16b9cbaf"
            },
            "id": "Data_Table_001",
            "externalIds": [
                {
                    "sourceName": "Test Data Table 001"
                }
            ],
            "className": "Data_Object",
            "name": "Data_Table_001",
            "description": "Description of Data Table 001",
            "slots": [
                {
                    "slotName": "contained_data_attributes",
                    "slotValue": [
                        "data_object_attribute_1",
                        "data_object_attribute_2",
                        "data_object_attribute_3",
                        "data_object_attribute_4"
                    ]
                }
            ]
        },
        ...
        {
            "_id": {
                "$oid": "5d0aaae2b225c42a16b9cbb3"
            },
            "id": "Data_Table_005",
            "externalIds": [
                {
                    "sourceName": "Test Data Table 005"
                }
            ],
            "className": "Data_Object",
            "name": "Data_Table_005",
            "description": "Description of Data Table 005",
            "slots": [
                {
                    "slotName": "contained_data_attributes",
                    "slotValue": [
                        "data_object_attribute_17",
                        "data_object_attribute_18",
                        "data_object_attribute_19",
                        "data_object_attribute_20"
                    ]
                }
            ]
        }
    ]
}
```

GET A SPECIFIC INSTANCE IN THE SPECIFIED COLLECTION

Use the following format to make a GET request for a specific instance (document) in a specific collection:

`https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}/instances/{instance-id}`

- {store-name} is the name of a NoSQL database instance. This should match the store-name used when POST-ing JSON content into the NoSQL database
- {collection-name} specifies which collection is to be queried, matching a collection that has been populated by a POST request
- {instance-id} matches the ID ('**_id**' in the NoSQL database) of the instance (document) in the specified collection

Example:

`https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-1/collections/data-object/instances/5d0aaae2b225c42a16b9cbaf`

returns HTTP 200 / OK with the following JSON body:


```
{
    "_id": {
        "$oid": "5d0aaae2b225c42a16b9cbaf"
    },
    "id": "Data_Table_001",
    "externalIds": [
        {
            "sourceName": "Test Data Table 001"
        }
    ],
    "className": "Data_Object",
    "name": "Data_Table_001",
    "description": "Description of Data Table 001",
    "slots": [
        {
            "slotName": "contained_data_attributes",
            "slotValue": [
                "data_object_attribute_1",
                "data_object_attribute_2",
                "data_object_attribute_3",
                "data_object_attribute_4"
            ]
        }
    ]
}
```


GET THE INSTANCES THAT ARE RELATED TO THE SPECIFIED INSTANCE VIA JSON ATTRIBUTE

This query can find instances (documents) that are related to other NoSQL instances in the specified store, or instances that exist within an Essential repository. This operation can query attributes that are either a JSON array or a JSON string value. Use the following format of request to query the NoSQL Database for instances (documents) that are related to the specified instance.

```
https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}/instances/{instance-id}/related-collections/{related-collection-name}/attributes/{attribute-of-related-collection}
```

- {store-name} is the name of a NoSQL database instance. This should match the store-name used when POST-ing JSON content into the NoSQL database
- {collection-name} specifies which collection is to be queried. This can match the name of a collection that has been populated by a POST request, or the name of a class in the Essential repository
- {instance-id} matches either the ID ('**_id**' in the NoSQL database) of the instance (document) in the specified collection in the NoSQL database, or the instance ID of an instance in the Essential repository
- {related-collection-name} specifies the collection that is to be queried for relations to the instance specified in {instance-id}
- {attribute-of-related-collection} specifies the attribute in the related collection that contains the identifier, or key, that matches the specified {instance-id}

This query structure can be interpreted as:

```
SELECT {related-collection-name} where {related-collection-name.attribute-of-related-collection} = {instance-id}
```

If the {instance-id} is contained in an attribute that is contained in a sub-object in the {related-collection-name} we can use 'dot' notation to specify which attribute is to be queried, e.g. `externalIds.sourceName`

Example: Find all instances of related data-object-attributes, that have 'parent-object' attribute matching "**5d0aaae2b225c42a16b9cbaf**".

```
https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-1/collections/data-object/instances/5d0aaae2b225c42a16b9cbaf/related-collections/data-object-attributes/attributes/parent-object
```

Returns HTTP 200 / OK with the following JSON body


```
{
    "instances": [
        {
            "_id": {
                "$oid": "5d091d8ab12fea717bc5b9cf"
            },
            "externalIds": [
                {
                    "sourceName": "Test Data Column 001"
                }
            ],
            "className": "Data_Object_Attribute",
            "name": "Data Column 001",
            "description": "Description of Data Column 001",
            "parent_object": "5d0aaae2b225c42a16b9cbaf",
```

```
            "slots": [
                {
                    "slotName": "belongs_to_data_object",
                    "slotValue": [
                        "Data_Table_001"
                    ]
                }
            ]
        }
        ...
        {
            "_id": {
                "$oid": "5d091d8ab12fea717bc5b9e7"
            },
            "externalIds": [
                {
                    "sourceName": "Test Data Column 025"
                }
            ],
            "className": "Data_Object_Attribute",
            "name": "Data Column 025",
            "description": "Description of Data Column 025",
            "parent_object": "5d0aaae2b225c42a16b9cbaf",
            "slots": [
                {
                    "slotName": "belongs_to_data_object",
                    "slotValue": [
                        "Data_Table_001"
                    ]
                }
            ]
        }
    ]
}
```


GET THE INSTANCES THAT ARE RELATED TO THE SPECIFIED INSTANCE VIA AN ESSENTIAL SLOTS ATTRIBUTE

This query can find instances (documents) that are related to other NoSQL instances in the specified store, or instances that exist within an Essential repository. This operation can query attributes that are either a JSON array or a JSON string value. Use the following format of request to query the NoSQL Database for all instances (documents) related to the specified instance via a 'slotValue' in the 'slots' array. This query searches slot arrays in the JSON of the specified type of instances (collection) for a match between the specified slotName and slotValue. The slotValue attribute is matched against all items within arrays (Note: the slotValue is an array of Strings in our Essential Utility API). Use the following format to make this query:

`https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}/instances/{instance-id}/related-collections/{related-collection-name}/slots/{slotName}`

- {store-name} is the name of a NoSQL database instance. This should match the store-name used when POST-ing JSON content into the NoSQL database

- {collection-name} specifies which collection is to be queried. This can match the name of a collection that has been populated by a POST request, or the name of a class in the Essential repository
- {instance-id} matches either the ID ('**_id**' in the NoSQL database) of the instance (document) in the specified collection in the NoSQL database, or the instance ID of an instance in the Essential repository
- {related-collection-name} specifies the collection that is to be queried for relations to the instance specified in {instance-id}
- {slotName} specifies the slotName value in a slots[] JSON array in the related collection that defines the slotName/slotValue contains the identifier (or key) that matches the specified {instance-id}

This query can be interpreted as:

```
SELECT {related-collection-name} where {related-collection-
name}.slots.{slotName} = {instance-id}
```

If the {instance-id} is contained in an attribute that is contained in a sub-object in the {related-collection-name} we can use 'dot' notation to specify which attribute is to be queried, e.g. `externalIds.sourceName`

Example: Find all instances of related data-object-attributes, that have a slotValue that matches 'Data_Table_001' in the slots[] array, when slotName equals 'belongs_to_data_object'.

```
https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-
1/collections/data-object/instances/5d0aaae2b225c42a16b9cbaf/related-
collections/data-object-attributes/slots/belongs_to_data_object
```

Returns HTTP 200 / OK with the following JSON body:


```
{
    "instances": [
        {
            "_id": {
                "$oid": "5d091d8ab12fea717bc5b9cf"
            },
            "externalIds": [
                {
                    "sourceName": "Test Data Column 001"
                }
            ],
            "className": "Data_Object_Attribute",
            "name": "Data Column 001",
            "description": "Description of Data Column 001",
            "parent_object": "5d0aaae2b225c42a16b9cbaf",
            "slots": [
                {
                    "slotName": "belongs_to_data_object",
                    "slotValue": [
                        "Data_Table_001"
                    ]
                }
            ]
        }
        ...
        {
            "_id": {
                "$oid": "5d091d8ab12fea717bc5b9e7"
            },
```

```
            "externalIds": [
                {
                    "sourceName": "Test Data Column 025"
                }
            ],
            "className": "Data_Object_Attribute",
            "name": "Data Column 025",
            "description": "Description of Data Column 025",
            "parent_object": "5d0aaae2b225c42a16b9cbaf",
            "slots": [
                {
                    "slotName": "belongs_to_data_object",
                    "slotValue": [
                        "Data_Table_005"
                    ]
                }
            ]
        }
    ]
}
```

POST A NEW DOCUMENT TO THE SPECIFIED COLLECTION IN THE SPECIFIED STORE

Use the following format to make a POST request that will create a new instance (document) in a specific collection:

`https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}`

- **stores** – which represent the databases in the NoSQL environment
- **collections** – which represent the collections in the NoSQL store (database)

Example:

`https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-1/collections/projects`

The JSON document that is to be posted to the specified collection can take any valid JSON form and must be included in a request body parameter, *json-file*. For example, the following document creates a new Project in the NoSQL database.

```
{
  "projectName":"Test Project 1",
  "projectDescription": "Test description 1"
}
```

On success, an HTTP / 201 response is returned with a JSON message that returns the newly created object in the collection. Note that the internal ID ("_id") of the new document is added to the JSON.

```
201 Created
```

```
{
    "_id": {"$oid": "5d2f2285f477926e8ad0652b"},
    "projectName": "Test Project 1",
```

```
    "projectDescription": "Test description 1"
}
```

If an error is encountered, an error HTTP response code with an error message will be returned.

- **400 – bad request**, indicating that there was a problem with the request that was received by the server, e.g. no JSON payload was supplied in the 'json-file' request body
- **500 – internal server error**, indicating that the server encountered a problem whilst servicing the request. Check your configuration and that all the elements of Essential Docker are running correctly

PUT AN UPDATE TO THE SPECIFIED DOCUMENT IN THE SPECIFIED COLLECTION IN THE SPECIFIED STORE

Use the following format to make a PUT request that will **update** the specified instance (document) in a specific collection:

```
https://<essential-docker-host>/api/essential-reference/v1/stores/{store-name}/collections/{collection-name}/instances/{instance-id}
```

- {store-name} is the name of a NoSQL database instance. This should match the store-name used when POST-ing JSON content into the NoSQL database
- {collection-name} specifies which collection is to be queried, matching a collection that has been populated by a POST request
- {instance-id} matches the ID ('**_id**' in the NoSQL database) of the instance (document) in the specified collection

Example:

```
https://<essential-docker-host>/api/essential-reference/v1/stores/tenant-1/collections/projects/instances/5d3072b9a21f8236a1273a5c
```

In addition to the above URL, the PUT operation must include a JSON document that represents the new, updated document in the specified collection. This document can take any valid JSON form and must be included in a request body parameter, *json-file*. For example, the following document updates a Project in the NoSQL database.

```
{
    "_id": {"$oid": "5d2f2285f477926e8ad0652b"},
    "projectName": "Test Project 1",
    "projectDescription": "Test description, updated",
    "projectDetails": "Newly added attribute"
}
```

Note that the "_id" attribute is optional for PUT requests

Returns HTTP 200 / OK with a JSON message that returns the newly updated object in the collection:

```
200 OK
```

```
{
    "_id": {"$oid": "5d2f2285f477926e8ad0652b"},
    "projectName": "Test Project 1",
    "projectDescription": "Test description, updated",
    "projectDetails": "Newly added attribute"
}
```


If an error is encountered, an error HTTP response code with an error message will be returned.

- **400 – bad request**, indicating that there was a problem with the request that was received by the server, e.g. no JSON payload was supplied in the 'json-file' request body
- **500 – internal server error**, indicating that the server encountered a problem whilst servicing the request. Check your configuration and that all the elements of Essential Docker are running correctly

APPENDIX 3 – SAML CONFIGURATION FOR ESSENTIAL DOCKER

SAML for Essential Docker allows customers to implement single sign-on for users. Users are provisioned using a just-in-time (JIT) approach with a default set of roles/permissions configured in the Essential Docker application. Once a user is provisioned all roles and permissions are managed directly in the Essential Docker application. Please note that single sign-out is not supported at this time.

PRE-REQUISITES

Please ensure you have set your hostname in the Platform Settings page before you attempt to configure SAML. Your hostname is used to generate the Single Sign-on URL (SSO URL) you will require for the SAML configuration. You can find instructions on how to set your hostname earlier in this document.

PART 1 – PLATFORM CONFIGURATION

On the host server running Essential Docker, edit the SAML configuration file using the following commands:

- **cd /eipdata/saml**
- **sudo nano application-authentication.xml**

Find the following section in the **application-authentication.xml** file and edit line 86, to change "**your-url.com**" to the URL of your directory server (e.g. ADFS), as highlighted below:

```
<AuthenticatorConfig name="SAMLSSOAuthenticator" enabled="true">

    <!--Add your custom hostname in the following parameter-->

    <Parameter name="SAMLSSOAssertionConsumerUrl">https://your-url.com/commonauth</Parameter>

    <!--Parameter name="SignAuth2SAMLUsingSuperTenant">true</Parameter-->

    <!--Parameter
name="SAML2SSOManager">org.wso2.carbon.identity.application.authenticator.samlsso.manager.Def
aultSAML2SSOManager</Parameter-->

</AuthenticatorConfig>
```

> *NOTE: It is critical to preserve the `/commonauth` part of the URL. For example, a valid URL would be* `https://ea.acmecorp.com/commonauth`

Once you have updated the URL, save the **application-authentication.xml** file.

Example:

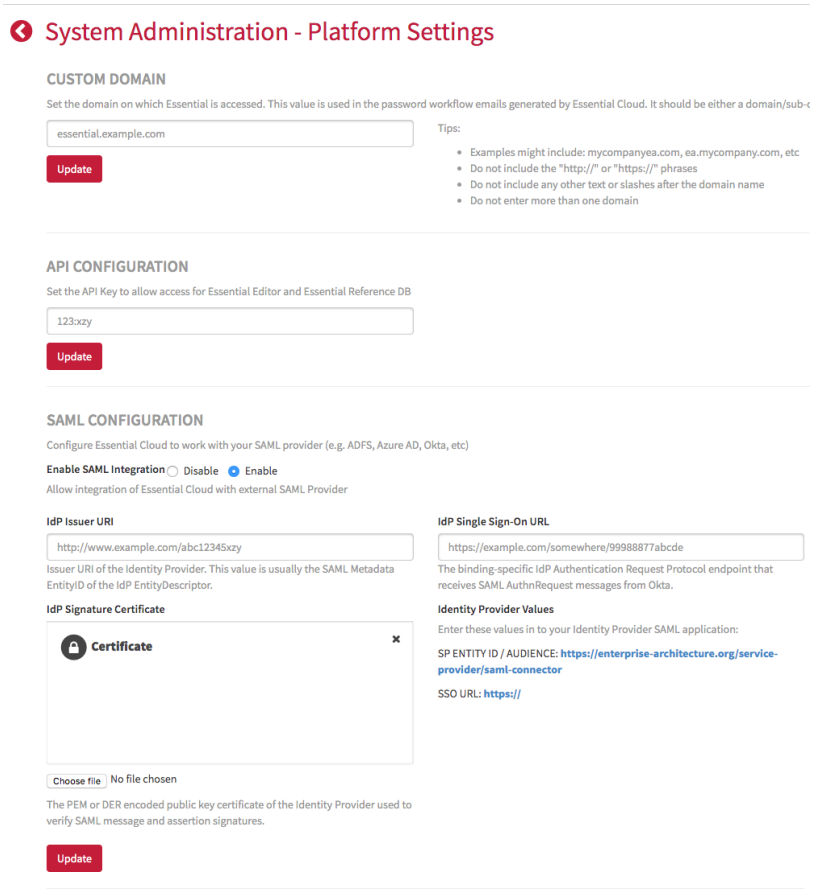
In the Nano editor, Ctrl+O saves the file, and Ctrl+X exits the Nano editor.

If Essential Docker is already running, you must restart the Essential Docker environment using the commands:

- **cd /essential**
- **sudo ./restart-essential-docker.sh**

PART 2- ESSENTIAL DOCKER CONFIGURATION

1. Log in to the Essential Docker application using an account with system administrator permissions.
2. Navigate to the System Administration option in the Configure menu.
3. You should then see a page listing the System Administration options, please select the Platform Settings option to show the following screen:



> *Ensure your hostname is displayed correctly in the Custom Domain section at the top of this page before proceeding any further.*

4. Make a record of the following information displayed on the right-hand-side of the SAML Configuration section:
    a. SP Entity ID / Audience
    b. SSO URL
5. Create a new SAML Application in your SAML Provider (e.g. Azure, Okta, etc) and add the ACS URL and Audience URI to the configuration
   Optional: we recommend creating a group for managing users rather than associating the whole directory.
6. Delete any default claims
7. Create new claims as described below for your specific SAML Provider
8. Collect the following information from the SAML application

a. Issuer URL (aka AD SAML Entity ID)
b. Single Sign-On URL (aka SAML 2.0 Endpoint)
c. X.509 Certificate (aka Certificate - base 64)

9. Add the values to the SAML configuration section in Essential Docker

CLAIMS – AZURE

CASE and **SPACING SENSITIVE**

Important: do not set any namespace declaration against the individual claims

Claim Name	Source attribute
email	user.mail
firstName	user.givenname
lastName	user.surname
Name ID	user.mail
username	user.mail

CLAIMS – ADFS

CASE and **SPACING SENSITIVE**

Important: do not set any namespace declaration against the individual claims

Claim Name	Source attribute
email	E-Mail-Addresses
firstName	Given-Name
lastName	Surname
Name ID	E-Mail-Addresses
username	E-Mail-Addresses

CLAIMS – PING PEDERATE

CASE and **SPACING SENSITIVE**

Important: do not set any namespace declaration against the individual claims

Claim Name	Source attribute
email	Email
firstName	First Name
lastName	Last Name
Name ID	Email
username	Email
SAML_SUBJECT	Email

CLAIMS – OKTA

CASE and **SPACING SENSITIVE**

Important: do not set any namespace declaration against the individual claims

Claim Name	Source attribute
email	user.email
firstName	user.firstName
lastName	user.lastName
Name ID	user.email
username	user.email

If you have any questions regarding SAML configuration then please contact support@enterprise-architecture.org

APPENDIX 4 – ADVANCED MEMORY CONFIGURATION

Customers can optionally configure the memory and some other parameters for some infrastructure components used within the container (e.g., Tomcat, MySQL, etc)

A new /eipdata/config/ folder is created with two files. One dedicated to MySQL and another for other components. A restart is required after changing these values.

-- eipdata
  -- config
    -- env.cnf
    -- custom-mysql.cnf

The **env.cnf** file contains configuration for the following components

- Tomcat (4 instances)
- Redis
- MongoDB
- Kafka

The **custom-mysql.cnf** contains a select set of configuration items for MariaDB (MySQL)

Changing these values can cause instability or the container to fail to start. We suggest you make a backup of this file before changing any values.

We recommend you do not set the memory values to more that 75% of the total available RAM as there are many other components that consume RAM which are not included in the configuration files.

ENV.CNF DEFAULTS

```
# Tomcat options for Import Utility (MB)
IMPORT_OPTS=-Xmx2048m

# Tomcat options for Viewer (MB)
VIEWER_OPTS=-Xmx6144m

# Tomcat options for API Platform (MB)
API_OPTS=-Xmx4096m

# Tomcat options for EDM (MB)
EDM_OPTS=-Xmx8192m

# Redis options (Bytes)
REDIS_OPTS=--maxmemory 2147483648

# MongoDB options (GB)
MONGODB_OPTS=--wiredTigerCacheSizeGB 1

# Kafka options (GB)
KAFKA_OPTS=-Xmx1G
```

CUSTOM-MYSQL.CNF DEFAULTS

```
[mysqld]
# InnoDB settings
innodb_buffer_pool_size=2G
max_allowed_packet=500M
innodb_lock_wait_timeout=30
max_connections=250
```