Installing an xUML Service Image

You can create a dedicated Docker image for single xUML services. To run an instance of the service, create a Docker container from such an image.
This guide describes the steps to perform to do this using **docker-compose**.

On this Page:

Related Documentation:

- License for Running xUML Services
- Docker compose

The Installation Process

Step 1: Preparations

Load the xUML Service Docker image with

```
docker image load -i xuml-<version>.tar
```

Step 2: Configure the Installation Settings

1. **Create a folder** to contain the xUML Service Docker configuration.

   > Please note that the docker-compose project is named after this folder. Also, the created Docker containers will get this folder name as a prefix.

2. **Collect the following files** to this folder:

   - the Dockerfile that comes with the xUML Service Docker image
   - for every service a valid xUML license
   - the xUML service repository of the service you want to deploy

3. **Create a file docker-compose.yml** to the previously created folder. This file should have the following content:

```
version: "3.3"
services:
  <your container name>:
    build:
      context: .
      args:
        REPOSITORY_FILE: <name of your xUML service repository file>
        XUML_IMAGE: <name of the previously deployed xUML image (see
step 1)>
    image: <your image name>:<your version>
    hostname: <your machine name>
    ports:
    - "<your external service control port>:<your Docker service
control port>"
    - "<your external service port>:<your Docker service port>"
```

Here, you can change the following settings:

Line	Setting	Description	Allowed Values / Example
3		Specify a container name prefix. This name will be appended with a consecutive number. The final name of your container will be: • `<Docker folder name (see step1)>_<container name prefix>_<a number>` • `xuml_hello_world_example_1`	`hello_world_example`

7	`REPOSITORY_FILE`	Specify the name of the xUML service repository file that should be deployed to the Docker container	`HelloWorldExample.rep`

8	`XUML_IMAGE`	Specify the name of the Docker image you have loaded in step 1.	`xuml:<version>`
			
			`xuml:2020.7`

9	`image`	Specify the name and version of the dedicated service image to be created from your settings.	`hello_world_example:1.0.0`
10	`hostname`	This hostname must match the machine name the xUML license is bound to.	`helloworld.scheer-acme.com`

12/13	`ports`	Map xUML service Docker container ports to the ports of the host.	`- "22020:22020"`
			
			`- "12020:12020"`

Step 3: Build the Service Image

Build the dedicated xUML service image with

```
docker-compose build
```

Step 4: Start the Container

Start the container by running the following command:

```
docker-compose up
```

To run the container in the background, use:

```
docker-compose up -d
```

You can stop the container using

```
docker-compose stop
```

Service Settings

xUML services can have settings that have been defined with the Builder. These settings are part of the service repository. They may have been set to default values on compile or may have been defined as to be provided by setting variables. Additionally, a service repository that has been exported from a Bridge may contain changed service settings.

When running an xUML service in an xUML service Docker container, you have the possibility to set these settings via environment variables of the container. To do this, add an **environment** section to the docker-compose file.

```
environment:
      - <setting name>=<setting value>
```

In this section you can assign values to settings. The name of the setting must be looked up in the service repository.

1. Unpack the service repository.
2. Open file **substitutions.xml**.
3. Lookup the setting you want to use in section **Variables**, e.g. `global_SomeIntialValues::Setting for a4`
4. Use the name provided by attribute **friendlyId** in your docker-compose file, e.g. `G_SOMEINTIALVALUES_SETTING_FOR_A4` and add `XUMLT_S_` as a prefix, e.g. `XUMLT_S_G_SOMEINTIALVALUES_SETTING_FOR_A4`

5. Provide a settings value, e.g. `XUMLT_S_G_SOMEINTIALVALUES_SETTING_FOR_A4=77`

Having changed only environment variables in your docker-compose file for an existing image, you do not need to rebuild the image. Just start the container, and the new values will be applied.

The Dockerfile Explained

The xUML service image comes with a **Dockerfile** that contains the necessary commands to combine the license and the repository to the dedicated xUML service image.

```
ARG REPOSITORY_FILE
ARG XUML_IMAGE

FROM $XUML_IMAGE

ARG REPOSITORY_FILE

COPY --chown=bridge:bridge license.xml $INSTANCES_HOME/license.xml
#COPY --chown=bridge:bridge logging.json $INSTANCES_HOME/logging.json

ADD $REPOSITORY_FILE /resources/repository.rep
```

Line	Variable/Command	Description	Allowed Values / Example
1	`REPOSITORY_FILE`	Defines the variable containing the name of the xUML service repository.	
2	`XUML_IMAGE`	Defines the variable containing the name of the base image.	
6	`ARG REPOSITORY_FILE`	Makes the variable visible to the service image.	
8	`COPY --chown=bridge:bridge license.xml...`	Copies the license file into the dedicated service image and applies the necessary user permissions.	

9	`COPY --chown=bridge:bridge logging.json...`	Copies the logging setup file to the dedicated service image and applies the necessary user permissions. This line is commented out. Uncomment this line if you want to change the logging format to other than JSON. In this case you need to provide a definition file for the logging format in the folder where you have stored the repository and the license file. For more information on how to change the logging format refer to xUML Runtime Logger Configuration.	
11	`ADD $REPOSITORY_FILE /resources/repository.rep`	Copies the repository file to the dedicated service image.	

Service startup is already integrated to the base image. Nevertheless, you can overwrite the startup command. To do this add an own command (`CMD`) to the Dockerfile, like e.g. `CMD /opt/xuml-tool /xuml-tool...`. For more details on the options of the xUML tool, refer to xUML Runtime Tool.