GSRS Docker Installation

**About this Document**

When implementing GSRS using Docker, you have the flexibility of either installing and running each GSRS microservice (a.k.a. “service” or “module”) in a Docker container under its own Tomcat server or having multiple services run in a single Docker container under the same Tomcat server. It is generally recommended to have one service in each Docker container to allow individual services to be assigned their own level of resources. That is how NCATS deploys GSRS. However, running everything in one Docker container is easier to set up and it can also serve as a starting point for GSRS docker deployment before you proceed to deploy each service into its own docker container.   
  
This document describes a strategy for a beginner Docker user to install the GSRS core functionality in Docker by using the repository of GSRS docker installation found at <https://github.com/epuzanov/gsrs3-docker/blob/main/README.md>. Following these instructions and steps, you can have a GSRS system including its core modules – the Frontend, Gateway and Substance modules – running in a Docker container for testing and exploration. Note that with the setup in the said Github repository, you can also install and run each service in its own docker container, but for that you will probably need to use the docker-compose.yml file rather than simply relying on the Dockerfile. Please refer to the repo for more details.

There are three sections to this document:

Section 1: Install a minimal version of GSRS using in-memory database H2

Section 2: How to configure a database other than H2

Section 3: How to create an image/container with an additional microservice

**Section 1: Install a minimal version of GSRS using in-memory database H2**

In this example, we use:

- Database for GSRS: H2

- Installation environment: Remote Linux server (20.04.1-Ubuntu).

**Steps:**

1. **Install Docker.**

You can find instructions online to install Docker. Below are two examples for reference.

<https://www.simplilearn.com/tutorials/docker-tutorial/how-to-install-docker-on-ubuntu>

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04>

We recommend that you first get familiar with basic Docker commands and concepts.

Here is the commands reference link:   
<https://docs.docker.com/engine/reference/commandline/docker/>

* Get familiar with docker commands to list, create, and delete Docker images/containers.
* Understand the use of a Dockerfile.
* Understand the docker compose file and how to start and stop the container using a compose file.
* Understand port mapping between a docker container and the host machine.

1. **Update the Dockerfile as needed.**

A Dockerfile for GSRS is located at:   
[https://github.com/epuzanov/gsrs3-docker/blob/main/Dockerfile](https://github.com/epuzanov/gsrs3-docker/blob/main/Dockerfile )

A Dockerfile is the only thing needed to Dockerize an application.

Modify these if you are testing with a different version than "3.1 public".

- ARG GSRS\_VER=3.1

- ARG GSRS\_TAG=GSRSv3.1PUB

- ARG EP\_EXT\_TAG=GSRSv3.1PUB

The argument in the docker file MODULE\_IGNORE (line 8) specifies which microservices to build.   
The complete set of available microservices includes adverse-events, applications, clinical-trials, frontend, gateway, impurities, products, ssg4m, invitro-pharmacology, and substances. You can list any microservices that you want to exclude from deployment. Alternatively, you can ignore the argument since it can be overridden in the building step too.

1. **Build the Docker image.**

Execute 'docker build' to create the docker image using the Dockerfile. The period at the end of the following command is to tell docker to build with the Dockerfile in the current directory. You will need to specify the file location with "-f" if the Dockerfile is not in the current directory or if it has a different name. We specify the MODULE\_IGNORE option to deploy only the code modules – frontend, gateway, and substances.

|  |
| --- |
| docker build --ulimit nofile=65535:65535 --build-arg MODULE\_IGNORE="adverse-events applications clinical-trials impurities products ssg4m" -t gsrs3:latest . |

Useful commands:

docker images : list the docker images

docker image rm <image-ID> : remove the image

1. **Run the image to start the Docker container.**

Execute the following command:

|  |
| --- |
| docker run -it -p 8080:8080 -v /var/lib/gsrs:/home/srs -e CATALINA\_OPTS='-Xms12g -Xmx12g -XX:ReservedCodeCacheSize=512m -Dgateway.allow.pattern="\d+\.\d+\.\d+\.\d+" -Ddeploy.ignore.pattern="(adverse-events|applications|clinical-trials|impurities|products|ssg4m)"' --name gsrs3-substances gsrs3:latest |

8080:8080 is the port mapping from host machine to the port in the container. If your local port 8080 is already in use and you want to use 9000, you can put 9000:8080 in the command.

The option --detach (or -d) can be used in the ‘docker run’ command to make it execute in the background.

This above ‘run’ command would start the substances backend, frontend, and gateway.

You can run a curl command to access the backend:

curl <http://localhost:8080/substances/api/v1/substances>

If this does not work for you, try from inside the docker container with (see docker exec).

If you are running this on your local machine, you can access the frontend in your browser too.  
http://frontend:8080/frontend/ginas/app/beta/home

Useful commands:

docker ps : list running containers

docker ps –a : list all containers

docker exec -it <container-ID> /bin/sh : connect to the container terminal

docker stop <container-ID> : stop the container

docker start <container-ID> : start the container

docker container rm <container-ID> : remove the container

1. **Configure Nginx for remote server.**

If you are working on a local machine, you can skip this part. If you are using a remote server, you might need to install and configure Nginx to make your application deployed in Docker accessible to remote users.

The installation of Nginx is out of scope of this document.

Below is the added section for GSRS in our Nginx config file for your reference.

|  |
| --- |
| server {  root /var/www/html;  # Add index.php to the list if you are using PHP  index index.html index.htm index.nginx-debian.html;  server\_name gsrs-test-public.ncats.io; # managed by Certbot  location / {  return 301 /ginas/app/beta;  }  location /ginas/app {  proxy\_http\_version 1.1; # this is essential for chunked responses to work  include proxy\_params;  client\_max\_body\_size 1000M;  # proxy\_pass http://localhost:9000;  proxy\_pass http://localhost:8080;  }  } |

**Section 2: How to configure a database other than H2**

You can use a different database **either**

1. by passing parameter variables when running the container, or
2. by changing configuration file values.

**a) Passing parameter variables when running the container**

In the above 'docker run' command (Section 1, Step 4 of this document), add these key values (presets) to point to an **external** Postgres database:

-e DB\_HOST='postgresql://db.server.org:5432/gsrsdb'.

-e DB\_USERNAME='postgres'

-e DB\_PASSWORD='yourpassword'

-e DB\_DDL\_AUTO='create' # or update or none

These values will be interpolated into microservice application.conf files and then assigned to Spring Boot datasource properties. (Note that accessing a database on the host system’s ‘localhost’ from inside the container will not work. There are ways to set up a database inside a docker network but that is beyond the scope of this document. This is why we specifically mention an **external** database.)

**b) Changing configuration file values**

To set the database in configuration files, we can take advantage of the fact that GSRS microservices’ application.conf files have an include directive at their bottom.   
  
For example, in the substances service the application.conf file has a line:

include conf/substances.conf

In the above 'docker run' command (Section 1, Step 4 of this document), notice that a **volume** is set with the option:   
 -v /var/lib/gsrs:/home/srs   
  
After running the container for the first time, GSRS-related files and folders will be created on the host system in the first folder in the above option. In /var/lib/gsrs/conf we can edit or create a conf/substances.conf file.   
  
Since the "include conf/substances.conf" directive is at the **bottom** of the application.conf file, we would set Spring Boot properties for the datasource in this way:

|  |
| --- |
| spring.datasource.driver-class-name="org.postgresql.Driver" spring.datasource.url="jdbc:postgresql://db.server.org:5432/gsrssubstances" spring.datasource.username="postgres" spring.datasource.password="yourpassword" spring.jpa.hibernate.ddl-auto="create" # first time, then change to update or none spring.jpa.database-platform="gsrs.repository.sql.dialect.GSRSPostgreSQLDialectCustom" spring.jpa.hibernate.use-new-id-generator-mappings="false" spring.hibernate.show-sql=true |

Once the configuration file has been edited and saved, return to the Linux command line to  
stop the container with 'docker container stop <container-ID>'   
and restart it with 'docker container stop <container-ID>'  
  
You can also access the configuration from inside the Docker container by accessing the files in the /home/srs folder with the command: docker exec –it <container-ID> /bin/sh

**Section 3: How to create an image/container for an additional microservice**

Let’s say we want to add the Products microservice to our docker image and container.   
  
If we assume that the Products module uses the same Postgres database as the Substances module set up above, and that we are using method (a) from above, we can rely on the same -e parameters. They will also be interpolated into the Product module’s application.conf file.

If we want to use the method “b” and we want Products to store its data in its own database, we should create a products.conf file in /var/lib/gsrs/conf. In it, we need to copy the same datasource properties as in the substances.conf file. Next, we add datasource properties specifically for Products. This is because the Products microservice makes use of the Substances datasource as well as its own datasource.

|  |
| --- |
| spring.datasource.driver-class-name="org.postgresql.Driver" spring.datasource.url="jdbc:postgresql://db.server.org:5432/gsrssubstances" spring.datasource.username="postgres" spring.datasource.password="yourpassword" spring.jpa.hibernate.ddl-auto="create" # first time, then change to update or none spring.jpa.database-platform="gsrs.repository.sql.dialect.GSRSPostgreSQLDialectCustom" spring.jpa.hibernate.use-new-id-generator-mappings="false" spring.hibernate.show-sql=true  product.datasource.driver-class-name="org.postgresql.Driver" product.datasource.url="jdbc:postgresql://db.server.org:5432/gsrsproducts" product.datasource.username="postgres" product.datasource.password="yourpassword" product.jpa.hibernate.ddl-auto="create" # first time, then change to update or none product.jpa.database-platform="gsrs.repository.sql.dialect.GSRSPostgreSQLDialectCustom" product.jpa.hibernate.use-new-id-generator-mappings="false" product.hibernate.show-sql=true |

Now we can build a new image and run a new container:

In the above 'docker build' command, remove 'products' from the MODULE\_IGNORE build argument.

Then, before you run the container from that image, edit the default deploy.ignore.pattern so that it does not ignore the Products module:

-Ddeploy.ignore.pattern="(adverse-events|applications|clinical-trials|impurities|ssg4m)'