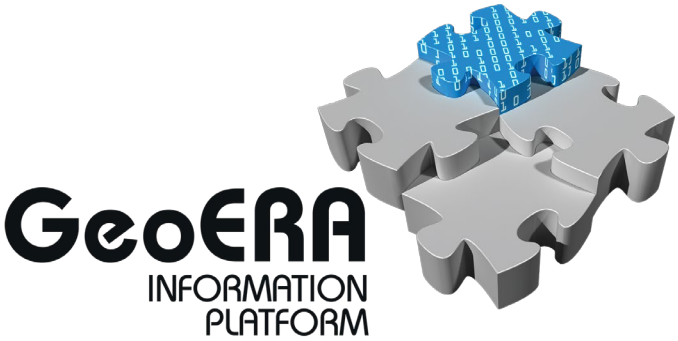

Deliverable 8.4

A series of example Docker containers

Authors and affiliation:
Marcus Sen, Sainath Chintham, Olufemi Ehindero, James Passmore
[BGS]

Bjørn Ove Grøtan
[NGU]

E-mail of lead author:
mase@bgs.ac.uk

Version: 28-10-2021

This report is part of a project that has received funding by the European Union's Horizon 2020 research and innovation programme under grant agreement number 731166.

Deliverable Data		
Deliverable number	D8.4	
Dissemination level	Public	
Deliverable name	A series of example Docker containers	
Work package	WP8, Data provider support	
Lead WP/Deliverable beneficiary	BGS	
Deliverable status		
Submitted (Author(s))	18/10/2021	Marcus Sen
Verified (WP leader)	18/10/2021	Patrick Bell
Approved (Coordinator)	29/10/2021	Jørgen Tulstrup

GENERAL INTRODUCTION

This deliverable has been modified from the original proposal as the primary method for scientific projects to deliver their data has become through the EDGI platform rather than setting up their own services. So, instead this report describes the potential benefits for data providers who do want to set up their own services of using container technology and the results of experiments done with containerising some example OGC web services.

TABLE OF CONTENTS

1 WHAT ARE CONTAINERS

In brief, containers are a way of packaging application software and dependencies into a standard file format that can then be deployed and run on a variety of generic computing platforms. These generic platforms can treat different containers similarly without having to know the details of the software in each container. A useful analogy is with shipping containers which enable the transport of a wide variety of cargo using the same handling machinery. This technology has become very popular recently as a way of deploying and flexibly running software services. We are not intending to provide a full or in-depth description of all the reasons for using containers here but will consider some possible benefits in the context of enabling scientific projects such as those in GeoERA to create web services providing their data. We will use Docker (https://www.docker.com/resources/what-container) for all the following examples, but it should be noted that there are now other software systems available for creating, managing and deploying container images in the same standardised format.

2 RATIONALE FOR USING CONTAINERS

We would like to enable data providers to set up and maintain standards based services to supply their data. Even if the final destination for a data set is the EGDI platform, a suitable container template could provide a convenient environment for testing and debugging configuration issues. One way this has been done in the past is to write detailed cookbooks that guide providers through how to install particular geospatial server software applications and configure them to use their data. For example, the OneGeology initiative provides instructions for data providers supplying geological map and other geoscientific data through Open GeoSpatial Commission standard web services using a variety of software at https://onegeology.github.io/documentation/providingdata.html. The Minerals4EU project provides documentation on putting data into a PostgreSQL database and serving it as an INSPIRE compliant WFS using deegree software at https://geusgitlab.geus.dk/m4eu. Even with detailed cookbooks it can be a challenge to produce a set of steps that can be followed by people with a wide range of technological expertise and that can work on or be easily adapted to a range of computing environments.

Container technologies may be able to make this easier in the following ways.

Although setting up the infrastructure to deploy containerised services is not easy, these are generic platforms that organisations are increasingly likely to have for all their application deployments. Thus, no special dedicated infrastructure needs to be set up for particular geospatial services if a generic container running platform already exists. If an organisation doesn't have such an infrastructure the services could be deployed on public cloud services or a shared common platform such as EGDI if it was set up as a container platform.

Many of the steps for the installation and configuration of particular server software can be automated in the container build scripts so that errors can be minimised. Dependencies can also be included so that customising the software installed on a dedicated server is not required. In many cases, updating server software may be as easy as pushing out an updated container definition which can be automatically deployed with minimal manual steps.

3 TECHNICAL TRIALS UNDERTAKEN

Production container services have not been set up as part of the GeoERA project but BGS and NGU have experimented with containerising typical services.

3.1 Docker

This is a minimal overview of Docker sufficient to understand the rest of this report.

Usually you will start with a base container image that someone else has made (you can create them from scratch but fewer people will need to do this). The base image commonly will contain a cut-down version of some Linux distribution[1] and may have some additional piece of application software installed and configured to run in some particular way. In the Docker ecosystem the canonical place to find images is Docker Hub (https://hub.docker.com) but there are other so-called "registries" where images can be found. Some of these are official images created by Docker or verified software publishers, others are shared by individuals who may have created them for their own purposes.

From this starting image you can customise the image with files and software you want to include in it using a recipe of instructions in a "Dockerfile". As well as adding files to the image you can run commands such as software installation or compilation commands in the same way you would on a machine running the same operating system as the base image.

The resulting Docker image can be run on a variety of platforms that have the generic ability to run containers without the need for specific software or libraries for the particular image. This could be a developer's machine for testing, an in-house server or server cluster or a public cloud service. There may need to be some specific configuration for a particular container where it needs to use shared resources outside itself. For example, mapping any network ports it uses to ones that are available on the host system. Importantly this may also cover file storage. Although a container has its own file system, once the container stops running, any changes made to files there are lost. The general paradigm of running containers is that they are considered immutable objects that can be started and stopped or replicated at will. Thus, if you have an application that needs to persist file system data there has to be some configuration to map parts of the containers filesystem to persistent storage on the host system. These kinds of specific configuration are, however, of a very generic nature and don't require anything like the level of individual tailoring or supporting software installation that would be required when installing application software directly onto a machine.

3.2 MapServer

MapServer (https://mapserver.org) is an Open Source platform for publishing spatial data and interactive mapping applications to the web. It supports several Open Geospatial Consortium (OGC) web service standards (https://mapserver.org/ogc/ogc_support.html#ogc-support). The OneGeology project has created documentation and example data and configuration files to provide Web Map Service (WMS), Web Feature Service (WFS) and Web Coverage Service (WCS)

[1] Windows containers also exist but are far less common and we have not looked at them.

using MapServer (https://onegeology.github.io/documentation/providingdata.html#using-mapserver). BGS has also created prototype WMS and WCS using some data from the GeoERA HOVER project that use MapServer (https://mapserver.org). For Task 8.4 BGS created a container version of these services.

There aren't official docker images for MapServer and the MapServer project itself doesn't publicise a docker image version[2]. For the purposes of this trial, without spending a lengthy amount of time comparing different ways to build an image, the published code (https://github.com/kartoza/docker-mapserver) from an established geospatial consultancy (https://www.kartoza.com/) was chosen to build the initial MapServer docker image. The code includes a Dockerfile (shown in the appendix) and various MapServer configurations and data files for copying into the docker image being created. After copying the files from the Kartoza repository their example MapServer docker image can be built on a developer machine with a simple one line command like:

```
docker build -t geoerams .
```

This creates a docker image called geoerams on the developer's machine which can be run with a command like:

```
docker run -d -p 8080:80 geoerams
```

The MapServer example service is then available for testing on the developer's machine at http://localhost:8080

In this particular example the Dockerfile is performing a number of steps to download MapServer source code and library dependencies and then compile the MapServer executable. This would be a complex series of instructions to give someone to follow manually but, as it is specified in the Dockerfile, they only need to execute the couple of commands above to have it running.

The next step is to configure the project specific services. MapServer works by having a number of text configuration files that are edited to configure the service and where the source data comes from. In the Dockerfile in the appendix the section starting with "# Start BGS configuration…" there are a number of lines that copy data and configuration files into the container image filesystem. These data and configuration files were taken from the pre-existing OneGeology and HOVER data services described above. With these changes, when docker build is run with the edited Dockerfile the OneGeology and HOVER services become available. However, while creating new services or modifying existing ones, it is convenient to be able to edit the configuration files while running the container to test the effect of any changes. This can be done by specifying a mapping of directories on the local filesystem to locations inside the container filesystem in the docker run command, e.g.:

```
docker run -d -v $PWD/ms/apps:/usr/local/src/ms/apps -p 8080:80 geoerams
```

[2] There are some images on docker hub at https://hub.docker.com/r/mapserver/mapserver with an unverified publisher http://www.mapserver.org but these have not been updated for 2 years. There are also some Dockerfiles in the MapServer source code but these have not been kept up to date with the latest version.

This means the configuration files in /ms/apps can be edited in the local filesystem and the effect seen on the running application. When the configuration is working correctly, another docker build command can be issued and the Dockerfile copies the same files into the container filesystem itself. From then we have an image containing all data and configuration that can be deployed as a single entity in different locations. In particular BGS have deployed it on their internal Kubernetes cluster and temporarily into an Amazon Elastic Cloud Services instance as described further on.

3.3 Deegree

NGU has undertaken work to containerise the deegree and PostGIS based technology stack that has been created for the Minerals4EU project.

They have looked at three different approaches to doing this.:

1. Reproduce the steps of the original GEUS cookbook in the Dockerfile, using the same war-files and configuration available at the GEUS repository.
2. Use official deegree war packages in the docker image build.
3. Use the official deegree docker images.

With each of these options, you have the possibility to separate your (preferably versioned) configuration and deegree-instances by using mounted volumes at runtime or by connecting to a separate PostgreSQL server (which could be a standalone database server or another containerised service itself with data files on mounted volumes).

Option 1 is good for education and intended for people used to setup/install manually on Windows using the M4EU cookbook, although once an experienced person has created the Dockerfile this may be re-usable by others with less experience.

Option 2 makes it easier to update deegree version numbers when/if security issues arises.

Option 3 is good for not having to maintain your own docker container, but you will have to live with some of the official degree image configuration such as a url endpoint other than <server>/m4eu/…/

3.4 GeoServer

BGS also carried out some experiments with reproducing some of the OneGeology GeoServer cookbook examples. One way of working with GeoServer is to edit text based configuration files, similarly to MapServer, however the configuration has to be reloaded after every change and the structure of the configuration files is not designed to be human friendly. Thus, the usual way of configuring GeoServer services is to use the web graphical interface to set things up on a running instance.

For the installation stage the manual method has various options from standalone packages to Java Servlet Container Web Application Archive (WAR) files that can be deployed in any of a number of available Java Application Servers. As it is a Java program there is not much difference between using Windows/Linux or other server operating systems once they have Java installed. As well as installing the main application, it is common to install extensions for various purposes such as accessing different kinds of data source or providing a greater variety of output formats etc. These

are generally just downloaded as collections of Java Archive (jar) files to be copied into the main application directories.

For this trial an example from another well-known geospatial consultancy (https://github.com/geosolutions-it/docker-geoserver) was used as a basis to start from. The Dockerfile is reproduced in the appendix. This system has been designed in a flexible way so that the docker image can be used itself to build new images using selected extensions and other options. BGS experimented with one service using an example GeoPackage data file from the OneGeology cookbook and one reproducing an existing borehole WFS that accessed data from an external Oracle database and made use of two optional GeoServer extensions.

The OneGeology example service could have been configured in a similar way to the MapServer example above on a developer machine. This would have involved mapping an external filesystem directory to the GeoServer data directory in the container. Instead of editing the configuration files directly as with MapServer, the web interface of GeoServer could then be used to configure the service and, when it was finished, a new docker build could be run to copy the modified data directory into a new container image. This would then be a single image deployable on any container platform.

This time, however, a different route was chosen where the basic GeoServer image was deployed to the BGS internal Kubernetes cluster but with a mapping of the GeoServer data directory to persistent file storage available to the cluster. This way, after configuring the service, the configuration persists on the cluster file system even if the container is restarted. The data file had to be copied to the persistent storage in a separate process.

The borehole WFS service was configured more in a similar way to the MapServer examples as we had an existing GeoServer data directory to copy from the production service. However, there were two notable additional conveniences afforded by the container build process. First, installing the two required optional extensions (app-schema and Oracle data store) simply needed their download locations specifying on a build command. Second, it turned out that this service wasn't compatible with the latest version of GeoServer. Using the ability to automatically generate a new version of the service using a different version of GeoServer simply by passing the version number to the build command made it easy to identify the version of GeoServer where the configuration broke. It would have been much more tedious to manually download different versions, configure and test them.

3.5 Deployment to Public Cloud

There are a number of different cloud service providers that can host containerised services so that once you have built an image or images you have a choice of deploying them in many places. Major ones include Amazon Elastic Container Service (https://aws.amazon.com/ecs/), Microsoft Azure (https://azure.microsoft.com/en-gb/services/kubernetes-service/docker/) and Google (https://cloud.google.com/) but there are many more.

For the purposes of this work, BGS took the above MapServer image, that had been deployed on a local development machine and on the BGS internal Kubernetes cluster already, and deployed it to an Amazon ECS instance. ECS has the capability to pull container images from private registries as well as public registries like Docker Hub. As

this was just a demonstration experiment a temporary machine address rather than a permanently addressed location was used but it illustrated that the same container image could be deployed in many locations. As with cloud-based services in general, AWS ECS deployments are priced on task size and resource usage so this must be taken into consideration.

4 CONCLUSIONS AND FUTURE WORK

We have carried out some experiments putting a variety of typical geospatial services into containerised applications. We have not settled on recommendations for the best practice for implementing services in this way but the work has suggested that this technology could make the process of setting up and deploying services easier for data providers in the future.

We have looked at putting the whole application in one or maybe two containers, that is, using a similar architecture to existing standalone deployments; simply packaged inside containers. We have not looked at major changes in the ways of delivering services such as moving to a microservice architecture. It is interesting to note that some moves are being made in that direction by existing geospatial software such as GeoServer being re-packaged for cloud native deployments (http://geoserver.org/geoserver-cloud/).

The following pattern of developing services for science data providers is suggested for simple cases without very large data sets. This pattern could be used, for example, in the proposed CSA successor project should that go ahead.

Data providers who want to set up their own services are given guidance on how to install Docker and Git on their own local development computers if they do not already have this.

Some basic template container build projects are made available either on an EGDI maintained Git repository or a public Git repository such as GitHub. These would enable building a basic example set of services using MapServer, GeoServer, deegree or other software with small example data set and configuration. The data provider is given some guidance by the support function on choosing one to use as a basis for their own service.

The data provider creates their own version controlled project using one of the templates and with the guidance of documentation and help desk adds their own data and configuration. During development and debugging of the service the project can be accessed from the central version control repository by both data provider and help desk staff so that they can test the running and collaboratively fix problems that may be found.

To deploy the final service the data provider may deploy on their own container platform if they have one. (Setting up a container platform is complex so would depend on the provider being capable of doing that themselves). If not then one option would be to use a public cloud offering, the other that could be considered would be to have a container platform added to EGDI so that providers could easily deploy the services they have configured there.

The above is a generic workflow and set of platforms that could work with smaller sized datasets supplied in some file format. If the data is to be obtained from an already existing database then creating an application container to access that would be

straightforward to configure but not so easy to collaboratively work on if the database isn't available from multiple locations. Working with large data sets would need some more thought on where to store the data.

A.1 EXAMPLE DOCKERFILES

These are just illustrative examples not intended for production use.

MapServer based on https://github.com/kartoza/docker-mapserver

```
# Mapserver for Docker ###
FROM ubuntu:focal
#If you change ubuntu version, don't forget to change 3 echo lines
MAINTAINER Admire Nyakudya<admire@kartoza.com>

ENV LANG C.UTF-8

# Update and upgrade system
RUN apt-get -qq update --fix-missing && apt-get -qq --yes upgrade

#-------------Application Specific Stuff ----------------------------------
----------------

# Install mapcache compilation prerequisites
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y software-properties-
common g++ make \
cmake wget git  bzip2 apache2 curl apache2-dev \
build-essential   openssl autoconf gtk-doc-tools libc-ares-dev libc-ares-
dev libpdf-api2-perl python3-pip \
swig protobuf-compiler python-setuptools libprotobuf-c-dev protobuf-c-
compiler libcurl4

# Install mapcache dependencies provided by Ubuntu repositories
RUN apt-get install -y --fix-missing --no-install-recommends \
    libxml2-dev \
    libxslt1-dev \
    libfribidi-dev \
    libcairo2-dev \
    librsvg2-dev \
    libmysqlclient-dev \
    libpq-dev \
    libcurl4-gnutls-dev \
    libexempi-dev \
    libfcgi-dev \
    libpsl-dev \
    libharfbuzz-dev \
    libexempi-dev \
    libgif-dev \
    libfcgi-dev \
    libjpeg62-dev \
    libproj-dev \
    libcairo2-dev \
    libprotobuf-dev \
    gdal-bin

RUN apt-get install -y libgdal-dev

# Install PHP7.4 and necessary modules
```

```
RUN  apt-get install -y php7.4-fpm libapache2-mod-php7.4 php7.4-
common php7.4-cli  php7.4 \
php7.4  php7.4-opcache  php7.4-gd php7.4-curl php7.4-fpm php7.4-dev php7.4-
mysql php7.4-mbstring  php7.4-xml

# Compile mapserver and associated resources
ADD resources /tmp/resources
ARG MAPSERVER_VERSION=branch-7-6
ADD setup.sh /setup.sh
RUN chmod 0755 /setup.sh
RUN /setup.sh

# Configure localhost in Apache
RUN cp  /tmp/resources/000-default.conf /etc/apache2/sites-available/
RUN wget http://mirrors.kernel.org/ubuntu/pool/multiverse/liba/libapache-mod-
fastcgi/libapache2-mod-fastcgi_2.4.7~0910052141-1.2_amd64.deb \
-O libapache2-mod-fastcgi.deb && dpkg -i libapache2-mod-
fastcgi.deb && apt install -f;rm libapache2-mod-fastcgi.deb

# Apache configuration for PHP-FPM # No fastcgi anymore
RUN cp /tmp/resources/php7-fpm.conf /etc/apache2/conf-available/

# Enable these Apache modules
RUN a2enmod actions  cgi alias proxy_fcgi  fastcgi headers
RUN a2enconf php7.4-fpm

# Link to cgi-bin executable
RUN chmod o+x /usr/local/bin/mapserv
RUN ln -s /usr/local/bin/mapserv /usr/lib/cgi-bin/mapserv
RUN chmod 755 /usr/lib/cgi-bin

EXPOSE  80
ENV DOCKERIZE_VERSION v0.6.1
RUN wget https://github.com/jwilder/dockerize/releases/download/$DOCKERIZE_VE
RSION/dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz \
    && tar -C /usr/local/bin -xzvf dockerize-linux-amd64-
$DOCKERIZE_VERSION.tar.gz \
    && rm dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz

#ifconfig not installed by default in focal
RUN apt-get install -y net-tools
ENV HOST_IP `ifconfig | grep inet | grep Mask:255.255.255.0 | cut -d ' ' -
f 12 | cut -d ':' -f 2`

# Fix php startup error https://stackoverflow.com/questions/59993170/php-7-4-
and-ubuntu-18-php-startup-unable-to-load-dynamic-library-curl-so
RUN mv /usr/local/lib/libcurl.so.4.4.0 /usr/local/lib/libcurl.so.4.4.0.backup

#RUN mkdir map
#RUN chmod -R 777 map
#COPY map map

#Start BGS Configuration for the map services and data files and logs .
```

```
#Cgi-bin scipts
RUN mkdir -p /usr/lib/cgi-bin/HOVER && mkdir -p /usr/lib/cgi-
bin/BGS_Bedrock_and_Superficial_Geology && mkdir -p /usr/lib/cgi-bin/oneg
COPY /cgi-bin/BGS_Bedrock_and_Superficial_Geology/wms /usr/lib/cgi-
bin/BGS_Bedrock_and_Superficial_Geology
COPY /cgi-bin/HOVER/wms /usr/lib/cgi-bin/HOVER
COPY /cgi-bin/oneg/wms /usr/lib/cgi-bin/oneg
RUN chmod 777 -R  /usr/lib/cgi-bin/*

#map files directories and moving

RUN mkdir -
p /usr/local/src/ms/apps/BGS_Bedrock_and_Superficial_Geology && mkdir -
p /usr/local/src/ms/apps/DefaultMapIncludes && mkdir -
p /usr/local/src/ms/apps/hover
COPY  /ms/apps/BGS_Bedrock_and_Superficial_Geology /usr/local/src/ms/apps/BGS
_Bedrock_and_Superficial_Geology/
COPY  /ms/apps/DefaultMapIncludes  /usr/local/src/ms/apps/DefaultMapIncludes/
COPY  /ms/apps/hover  /usr/local/src/ms/apps/hover/
RUN chmod 655 -R /usr/local/src/ms/apps/*

#Httpd.d configurations
RUN mkdir -p /usr/local/src/ms/httpd.d
COPY  /ms/httpd.d/* /usr/local/src/ms/httpd.d/
RUN chmod 777 -R /usr/local/src/ms/httpd.d/*

#logs directories
RUN mkdir -p /usr/local/src/ms/logs/BSG && mkdir -
p /usr/local/src/ms/logs/hov && mkdir -
p /usr/local/src/ms/apps/OUT && mkdir -p /usr/local/src/ms/out
RUN touch /usr/local/src/ms/logs/BSG/ms_error.log && touch /usr/local/src/ms/
logs/hov/err.log
RUN chmod 777 -R /usr/local/src/ms/logs/BSG/ms_error.log && chmod 777 -
R /usr/local/src/ms/logs/hov/err.log
RUN chmod 777 -R /usr/local/src/ms/* && chmod 777 -R /usr/local/src/ms/out/

# add path of httpd.d to the apche2 conf
RUN echo "Include /usr/local/src/ms/httpd.d/httpd_*.conf" >> /etc/apache2/apa
che2.conf

# end of BGS configuration

RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

CMD ["dockerize", "-stdout", "/var/log/apache2/access.log", "-
stderr", "/var/log/apache2/error.log", "apachectl", "-D", "FOREGROUND"]
```

GeoServer based on https://github.com/geosolutions-it/docker-geoserver

```
FROM tomcat:9-jdk11-openjdk as mother
LABEL maintainer="Alessandro Parma <alessandro.parma@geosolutionsgroup.com>"
SHELL ["/bin/bash", "-c"]
```

```
# download and install libjpeg-2.0.6 from sources.
ARG DEBIAN_FRONTEND=noninteractive
ARG CMAKE_BUILD_PARALLEL_LEVEL=8
ARG APP_LOCATION="geoserver"
RUN apt-get update && apt-get install -y unzip wget cmake nasm\
    && wget https://nav.dl.sourceforge.net/project/libjpeg-
turbo/2.0.6/libjpeg-turbo-2.0.6.tar.gz \
    && tar -zxf ./libjpeg-turbo-2.0.6.tar.gz \
    && cd libjpeg-turbo-2.0.6 && cmake -G"Unix Makefiles" && make deb \
    && dpkg -i ./libjpeg*.deb && apt-get -f install \
    && apt-get -y purge cmake nasm\
    && apt-get clean \
    && apt-get -y autoclean \
    && apt-get -y autoremove \
    && rm -rf /var/lib/apt/lists/* \
    && rm -rf /usr/share/man/* \
    && rm -rf /usr/share/doc/*

# accepts local files and URLs. Tar(s) are automatically extracted
WORKDIR /output/datadir
ARG GEOSERVER_DATA_DIR_SRC="./.placeholder"
ADD "${GEOSERVER_DATA_DIR_SRC}" "./"

# accepts local files and URLs. Tar(s) are automatically extracted
WORKDIR /output/webapp
ARG GEOSERVER_WEBAPP_SRC="./.placeholder"
ADD "${GEOSERVER_WEBAPP_SRC}" "./"

# zip files require explicit extracion
RUN \
    if [ -f "./download" ] ; then \
      mv download geoserver.war.zip && unzip geoserver.war.zip -
d geoserver.war && mkdir -
p ./geoserver && unzip ./geoserver.war/geoserver.war -d ./geoserver && rm -
rf ./geoserver.war;\
    fi

# zip files require explicit extracion
RUN \
    if [ "${GEOSERVER_WEBAPP_SRC##*.}" = "zip" ]; then \
        unzip "./*zip"; \
        rm ./*zip; \
    fi \
    && [ -d "./geoserver" ] || (mkdir -
p ./geoserver && unzip ./geoserver.war -d ./geoserver && rm ./geoserver.war)

WORKDIR /output/plugins
ARG PLUG_IN_URLS=""
ARG PLUG_IN_PATHS=""
ADD .placeholder ${PLUG_IN_PATHS} /output/plugins/
COPY geoserver-plugin-download.sh /usr/local/bin/geoserver-plugin-download.sh
```

```
RUN /usr/local/bin/geoserver-plugin-
download.sh /output/plugins/ ${PLUG_IN_URLS}
RUN \
    if [ -f *.zip ] ; then \
       unzip -o "./*.zip"; \
    fi

WORKDIR /output/webapp
RUN \
    if [ "${APP_LOCATION}" != "geoserver" ]; then \
      mv /output/webapp/geoserver /output/webapp/${APP_LOCATION}; \
    fi

FROM tomcat:9-jdk11-openjdk

ARG UID=1000
ARG GID=1000
ARG UNAME=tomcat
ARG CUSTOM_FONTS=""
ENV ADMIN_PASSWORD=""
ENV APP_LOCATION="geoserver"

ENV CATALINA_BASE "$CATALINA_HOME"
# set externalizations
ENV GEOSERVER_HOME="/var/geoserver"
ENV GEOSERVER_LOG_DIR="${GEOSERVER_HOME}/logs"
ENV GEOSERVER_DATA_DIR="${GEOSERVER_HOME}/datadir"
ENV GEOSERVER_LOG_LOCATION="${GEOSERVER_LOG_DIR}/geoserver.log"
ENV GEOWEBCACHE_CONFIG_DIR="${GEOSERVER_DATA_DIR}/gwc"
ENV GEOWEBCACHE_CACHE_DIR="${GEOSERVER_HOME}/gwc_cache_dir"
ENV NETCDF_DATA_DIR="${GEOSERVER_HOME}/netcdf_data_dir"
ENV GRIB_CACHE_DIR="${GEOSERVER_HOME}/grib_cache_dir"
# override at run time as needed JAVA_OPTS
ENV INITIAL_MEMORY="2G"
ENV MAXIMUM_MEMORY="4G"
ENV LD_LIBRARY_PATH="/opt/libjpeg-turbo/lib64"
ENV JAIEXT_ENABLED="true"
ENV PLUGIN_DYNAMIC_URLS=""
ENV GEOSERVER_OPTS=" \
  -Dorg.geotools.coverage.jaiext.enabled=${JAIEXT_ENABLED} \
  -Duser.timezone=UTC \
  -Dorg.geotools.shapefile.datetime=true \
  -DGEOSERVER_LOG_LOCATION=${GEOSERVER_LOG_LOCATION} \
  -DGEOWEBCACHE_CONFIG_DIR=${GEOWEBCACHE_CONFIG_DIR} \
  -DGEOWEBCACHE_CACHE_DIR=${GEOWEBCACHE_CACHE_DIR} \
  -DNETCDF_DATA_DIR=${NETCDF_DATA_DIR} \
  -DGRIB_CACHE_DIR=${GRIB_CACHE_DIR}"

ENV JAVA_OPTS="-Xms${INITIAL_MEMORY} -Xmx${MAXIMUM_MEMORY} \
  -Djava.awt.headless=true -server \
  -Dfile.encoding=UTF8 \
  -Djavax.servlet.request.encoding=UTF-8 \
  -Djavax.servlet.response.encoding=UTF-8 \
```

```
  -XX:SoftRefLRUPolicyMSPerMB=36000 -XX:+UseG1GC \
  -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 \
  ${GEOSERVER_OPTS}"

COPY run_tests.sh /docker/tests/run_tests.sh

# install needed packages and create externalized dirs
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update \
    && apt-get install --yes git gdal-bin postgresql-
client fontconfig libfreetype6 jq \
    && apt-get clean \
    && apt-get -y autoclean \
    && apt-get -y autoremove \
    && rm -rf /var/lib/apt/lists/* \
    && rm -rf /usr/share/man/* \
    && rm -rf /usr/share/doc/* \
    && mkdir -p \
    "${GEOSERVER_DATA_DIR}" \
    "${GEOSERVER_LOG_DIR}"  \
    "${GEOWEBCACHE_CONFIG_DIR}" \
    "${GEOWEBCACHE_CACHE_DIR}" \
    "${NETCDF_DATA_DIR}" \
    "${GRIB_CACHE_DIR}"

# copy from mother
COPY --from=mother "/opt/libjpeg-turbo" "/opt/libjpeg-turbo"
COPY --from=mother "/output/datadir" "${GEOSERVER_DATA_DIR}"
COPY --
from=mother "/output/webapp/geoserver" "${CATALINA_BASE}/webapps/geoserver"
COPY --from=mother "/output/plugins" "${CATALINA_BASE}/webapps/geoserver/WEB-
INF/lib"

COPY geoserver-plugin-download.sh /usr/local/bin/geoserver-plugin-download.sh
COPY geoserver-rest-config.sh /usr/local/bin/geoserver-rest-config.sh
COPY geoserver-rest-reload.sh /usr/local/bin/geoserver-rest-reload.sh
COPY entrypoint.sh /entrypoint.sh
COPY ${CUSTOM_FONTS} $GEOSERVER_DATA_DIR/styles
RUN groupadd -g $GID $UNAME
RUN useradd -m -u $UID -g $GID --system $UNAME
RUN chown -
R $UID:$GID $GEOSERVER_LOG_DIR $CATALINA_BASE $GEOWEBCACHE_CACHE_DIR $GEOWEBC
ACHE_CONFIG_DIR $NETCDF_DATA_DIR $GRIB_CACHE_DIR $GEOSERVER_DATA_DIR

RUN if [ ! -f "${GEOSERVER_DATA_DIR}/logging.xml" ]; then cp -
a ${CATALINA_BASE}/webapps/geoserver/data/* ${GEOSERVER_DATA_DIR};fi

WORKDIR "$CATALINA_BASE"
USER $UNAME

ENV TERM xterm
EXPOSE 8080/tcp
CMD ["/entrypoint.sh"]
```