**Dockerize a Shiny Application**

1. Create a dockerfile that contains all of the instructions on how to run your shiny application

# Get shiny+tidyverse packages from rocker image

FROM rocker/shiny-verse:4.0.3

# Set up a volume directory in the docker image

VOLUME /srv/shiny-server/

# Set up a working directory in the docker image

WORKDIR /srv/shiny-server/

# Install system libraries of general use

RUN apt-get update && apt-get install -y \

libudunits2-dev \

libv8-dev \

libsodium-dev

# Install the required R packages to run the app

RUN R -e "install.packages('shinydashboard', dependencies=TRUE, repos='http://cran.rstudio.com/')"

RUN R -e "install.packages('shinythemes', dependencies=TRUE, repos='http://cran.rstudio.com/')"

RUN R -e "install.packages('shiny', dependencies=TRUE, repos='http://cran.rstudio.com/')"

RUN R -e "install.packages('DT', dependencies=TRUE, repos='http://cran.rstudio.com/')"

# Expose ShinyApp to a local port

EXPOSE 3838

# Copy bash file that runs the shiny application to Docker image

COPY shiny-server.sh /usr/bin/shiny-server.sh

# Allow permission the files or directory

RUN ["chmod", "+rwx", "/srv/shiny-server/"]

RUN ["chmod", "+x", "/usr/bin/shiny-server.sh"]

# Excute the bash file to run the app in the docker image

CMD ["/usr/bin/shiny-server.sh"]

1. Navigate to where the dockerfile is stored and run the following commands to build a docker image for your application

cd "Documents/Lab Meeting/"

docker build -t simple-app:0.0.1 .

*-t is used to tag the image such as the version of the application*

1. To check if an image is built successfully,

docker images

1. Run the following command to build a docker container and published the app on the host machine

docker run -d -p 9898:3838 --name shiny-app \

-v "/Users/reinachau/Documents/Lab Meeting/shinyApps":/srv/shiny-server/ \

-v "/Users/reinachau/Documents/Lab Meeting/shinyApps-log/":/var/log/shiny-server/ \

simple-app:0.0.1

-*d is to run the container in detached mode*

*-name is to assign a name to the container  
-p is to link the container port onto the host port and publish the application through the host domain <host\_port>:<container\_port>*

*-v is to mount the codebase or data files from the host directory onto the container directory <host\_dir>:<container\_dir>  
  
For more information about the syntax usage, see*[***docker documentation***](https://docs.docker.com/engine/reference/builder/)

How Shiny talks to Docker

Shiny → listen on localhost (127.0.0.1 or 0.0.0.0) → published or exposed the application on port 3838/8787/4848/…

Docker → listens on the exposed port → publish the application on host port 80/443/8080/3838/8000/…

1. To check if a container is built successfully

docker container ps

1. Once the app is up and running, you can visit the local host, http://localhost:8989 to verify if the application is indeed hosted there
2. You can share your image on [Docker Hub](https://hub.docker.com/). Before pushing the image to Docker Hub, you must tag the image based on the name of your user account.

docker tag simple-app:0.0.1 montibot/simple-app:0.0.1

docker login --username=montibot --password=notsosecret9

docker images

docker push montibot/simple-app:0.0.1

**NGINX and Docker-Compose**

**NGINX** is an open source web server and reverse proxy technology used for hosting websites and applications.

**Docker-Compose** is a tool for defining and running multi-container Docker applications. We use a YAML file to configure the application’s service.

Here are the seven steps to set-up NGINX and docker-compose:

1. Install docker-compose
2. Build a NGINX image from Docker Hub without installing NGINX software
3. Dockerize the shiny application locally or pull the image from Docker Hub
4. Configure NGINX configuration files
5. Create a docker-compose.yml file
6. Run the docker-compose.yml file
7. Check to see if the application is running on http or https

**Step 1**: Install docker-compose

See how to install Docker-compose [**here**](https://docs.docker.com/compose/install/)

To check if docker-compose is installed,

docker-compose --version

**Step 2**: Build a docker image for NGINX

An image for NGINX can be found at [**Docker Hub**](https://hub.docker.com/_/nginx). You can pull this image without the need to install the NGINX software locally,

docker pull nginx:latest

To check if the image is built successfully,

docker images

Also see this [**link**](https://hub.docker.com/_/nginx) on how to use the image

**Step 3**: Dockerize the shiny application locally or pull the image from Docker Hub

docker pull simple-app:0.0.1

**Step 4**: Configure NGINX configuration files

There are two configuration files for NGINX, nginx.conf and default.conf.template files.

The nginx.conf file contains the standard configuration for NGINX. Unless you know how to add directives specified for a configuration file, otherwise, I do not recommend making any changes to this file.

The default.conf.templat file contains configuration that allows NGINX to direct any encrypt or unencrypt traffics to an application that is published on a specific port on the host machine.

How NGINX talks to Docker

With NGINX,

NGINX → listens on → port 80 (http) and 443 (https)

With Docker,

Docker → listens on → port 80/443/8080/3838/8000/…

With SHINY

Shiny → listens on localhost → port 3838/8787/4848/…

In other words, when we run a docker container, we basically expose our application through a port that can be viewable thru a Shiny's localhost. Docker listens to this port and will publish the application to a specified port on the host machine (for example: port 7856 for simplicity). Therefore, when we navigate to port 7856 on the host domain (for example http://[domain\_name/ip address]:7856), we will see that the shiny app is hosted there.

Nevertheless, we probably don't want to navigate to the application via a port link. Thus, we can use the NGINX configuration files to create an alias location for the application (for example http://[domain\_name/ip address]/simple-app/). When a user is navigating to that specific address on the host domain, NGINX will transfer that traffic to the port 7856 which the application was originally published on. As the result, NGINX is served as a reverse proxy for hosting our applications.

Here is a snapshot of how the http configuration file looks like:

server {

listen 80;

server\_name montilab.bu.edu; #ip address or domain name

server\_tokens off;

# Load configuration files for the default server block.

include /etc/nginx/default.d/\*.conf;

# Specific the location for the simple app

location /simple-app/ {

rewrite ^/ simple-app/(.\*)$ /$1 break;

proxy\_pass http://montilab.bu.edu:7856/;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection $connection\_upgrade;

proxy\_read\_timeout 20d;

proxy\_buffering off;

}

}

**Step 5**: Create a docker-compose.yml file

Once we had configured NGINX as a reverse proxy for our application, we can create a docker-compose.yml file that comprises of all the containers that we want to communicate with the NGINX container.

To enable communication between containers, in the docker-compose file, we must create a service to run the NGINX container and a service to run the container for the simple-app application. Under each service, we need to specify a list of instructions of how the container can be built, for instance, what image is used to build the container, what do we want to name the container, what port is used to expose the shiny app to Docker, and what port is used to publish the app onto the host machine.

After we had defined all of these keys components of how the containers are built, in the NGINX configuration files, we need to make sure that the host port matches the port that NGINX redirect the encrypt or unencrypt traffic to the application.

Here is a snapshot of the docker-compose.yml file:

version: '3.6'

services:

**nginx**:

image: nginx:latest #name of the image

container\_name: nginx #name of the container you want to build

restart: always #allow the connection to restart if it gets disconnected

ports:

- 80:80 #specify the unencrypt port

- 443:443 #specify the encrypt port but will need ssl later

volumes:

- /home/docker/nginx/nginx.conf:/etc/nginx/nginx.conf #connect the host configuration files to docker config files

- /home/docker/nginx/conf.d:/etc/nginx/templates #connect the host configuration files to docker config files

command: [nginx-debug, '-g', 'daemon off;']

**simple-app**:

image: montibot/simple-app:0.0.1 #name of the docker image

container\_name: simple-app #name of the container

restart: always. #allow the connection to restart if it gets disconnected

ports:

- 7856:7856 #port where the application is published on

volumes:

- /home/xposome/shinyApps/:/srv/shiny-server/ #location to the codebse or data files

- /home/xposome/shinyApps-log/:/var/log/shiny-server/ #location to store log information from the shiny app

**Step 6**: Run the docker-compose file

After we have the docker-compose file and NGINX configuration files all set up, we can cd to where docker-compose.yml is located and run the following command on the terminal.

docker-compose up -d

*-d is used to run docker compose in detach mode*

**Step 7**: Check to make sure the application is running on http or https

Navigate to the pre-specified domain (for example **https://montilab.bu.edu/simple-app/**) on host server and see if the application is hosted there.