Article
Patrick Jamieson · Apr 22, 2019 8m read

Launching IRIS Using Docker

Launching IRIS Using Docker

This brief document will walk through the steps to launch IRIS community edition using Docker on the MAC. For those new to Docker, the first two pages explain how to install Docker, and run some basic commands that will be needed to get the IRIS community edition running. Experienced Docker users can skip directly to page three.

1. Get Docker Desktop for MAC. This can be downloaded at Docker Hub, but one should look at the requirements before downloading it to ensure it works smoothly. The directions and download link are at: https://docs.docker.com/docker-for-mac/install/ Windows installations of Docker require Docker Toolbox, because Docker cannot run natively on Windows. Docker Toolbox creates a virtual machine that Docker runs in for Windows users. Windows users should follow the directions on the Docker website: https://hub.docker.com/editions/community/docker-ce-desktop-windows

2. Experiment with Docker. The first logical command is to determine what docker images are in the local repository. Using the Mac terminal type: **docker images**. At this stage nothing will be found, but one will see the following headers: Repository, Tag, Image ID, Created, and Size. Now, run a test container by specifying the following at the command line: **docker run busybox:1.24 echo " hello world "** Since the busybox image was not found in the docker repository it was " pulled " from docker hub, and the echo command was executed in the container, printing out " hello world " at the command line. Now try executing **docker images** again, you should now see the busybox image in the local repository. It is important when using Docker to specify both the name of the image and the tag, separating them by a colon. This lets docker know which version of the image to run (e.g. **busybox:1.24**). When a container is present locally, it will not be downloaded again.

3. Try running another command on the Docker image. Type **docker run busybox:1.24 ls /**. One should see a directory listing of directories and files at the root of the container.

4. There are two important options when running a container. The -I flag starts an interactive container, and the -t flag creates a pseudo-TTY that attaches stdin and stdout. Type this: **docker run -i -t busybox:1.24**. One can now run commands in the container shell, including adding files. Typing **exit**, will exit the container and shut it down. Any files created while in the container will be lost.

5. Containers are normally run in the background (not interactively). The -d flag, which stands for detach runs the container in " detached " mode. Type this command **docker run -d busybox:1.24 sleep 1000**. Docker returns immediately the container id, and the container is running. To verify this type **docker ps**. This will show what containers are running, which in our case is this single container. To see containers that are running and have previously run type **docker ps -a**. To remove a container when it exits type **docker run –rm busybox:1.23 sleep 1**. The container will execute but will not be found in the history list using docker ps -a.

6. Docker generates funny container " names " when running containers, we can specify the name we want Docker to generate using the –name option. Type **docker run –-name mycontainer busybox:1.24**. Now when using **docker ps -a**, one will see then name of the container is mycontainer.

7. Docker provides a handy function for getting details on low level information on containers such as IPAddress and Ports. Type **docker inspect containerId** (replace containerId with one the container ids from docker ps -a).

8. Frequently containers run processes that " listen " on a specific port. In order to communicate to the container port from the host, one must map the host port to the container. The format of the command is -p hostport: containerport. If a docker container is running locally, the container IP address is " localhost ". On windows it may be 127.0.0.1. A useful command for troubleshooting is it to check the logfile of the container. This can be easily done using the command **docker logs containerId** (replace containerId with one the container ids from docker ps -a)

9. Containers are actually made up of layers. Starting with the base layer, additional layers are added. Docker provides the **history** command to list the layers in a container. Type **docker history busybox:1.24**

10. There is much more that docker can do, including building more complex images using the commit command or a Dockerfile (refer to: https://docs.docker.com/). This brief intro on Docker provides the basics for getting the IRIS community container up and running.

Launching the IRIS Container

1. The instructions for using the Docker container for the IRIS Community Edition are found at: https://hub.docker.com//intersystems-iris-data-platform/plans/222f869e-567c-4928-b572-eb6a29706fbd?tab=instructions

2. One additional Docker command must be explained to configure the password file. Docker containers cannot access the host file system directly. The container must mount a directory made available to it from the host. Initially, Docker for Mac makes the following directories available to be bind/mount into Docker containers: /Users, /Volumes, /private, /tmp. To store passwords outside the container (remember all files written in the container will disappear on exit), we need to create a file in /Users/yourUserName/external/password.txt. Mounting a volume uses the flag -v and then the hostname followed by a colon and then the container directory.

3. Now run the container we the following command: **docker run --name irisdb -v /external:/external -d store/intersystems/iris:2019.1.0.511.0-community --password-file /external/password.txt.** After running this command and checking the status with the command **docker ps -a**, you should see the following after a couple of minutes:

CONTAINER ID IMAGE COMMAND CREATED
STATUS PORTS NAMES

d74f2b7a48d5 store/intersystems/iris:2019.1.0.511.0-community "/iris-main --passwo…"
2 minutes ago Up 2 minutes (healthy) irisdb

4. Connect to the container just launched with the command: docker exec -it irisdb iris session iris. This connection will allow us to login to an iris session. The Username is **SuperUser** and the Password is **SYS**. IRIS will now ask that the password be changed, as shown below:

Node: d74f2b7a48d5, Instance: IRIS

Username: SuperUser

Password: ***

Password change required

Please enter a new password

Password: *********

Please retype your new password

Password: *********

USER>

Now we can type commands directly at the IRIS terminal command prompt. Such as:

USER>set firstname = "Peter"

USER>write firstname

Peter

USER>

We can type interactive commands, but what about launching the management portal?

Normally, to do this we would open a browser and typing the link: http://localhost:52773/csp/sys/UtilHome.csp , but when we do this an error message is displayed, and the localhost site refuses to connect. Remember the section on port mapping and how containers are isolated from the external network without them? We must map two

ports, the IRIS superserver default port at 51773, and the management portal port at 52773.

In the above case, first stop the command line process by typing **Halt**. Next stop the container by typing: **docker stop [containerid]**

5. Let ' s start up this container again with port mapping**docker run -p 51773:51773 -p 52773:52773 -v /external:/external -d store/intersystems/iris:2019.1.0.511.0-community --password-file /external/password.txt**

Now opening a browser window should with the link http://localhost:52773/csp/sys/UtilHome.csp , should reveal the management portal.

6. Even though we managed to use the password file on the host system, we were not able to persist the password change we made earlier. Why not? IRIS must be must be made aware of the directory of where to store it persistent configuration settings. We must add a flag --env ISCDATADIRECTORY=/external/iconfig.

Thus, the full command reads: **docker run -p 51773:51773 -p 52773:52773 -v /external:/external --env ISCDATADIRECTORY=/external/ifconfig -d store/intersystems/iris:2019.1.0.511.0-community --password-file /external/password.txt**

7. At this point, you should not need to change the password on the management portal that you set initially.

8. The command string is difficult to remember, and another approach is to store it in a file that a related docker utility called docker-compose employs. The file is of type yaml or yml, and must be named docker-compose. The docker-compose file offers numerous ways to configure any number of containers and their dependencies. However, for our purposes a simpler file is all that is necessary. Here are the contents (the file must be labeled **docker-compose.yml**):

```
version: '3'

services:

  iris:

   image: store/intersystems/iris:2019.1.0.511.0-community

   ports:

    - "51773:51773"

    - "52773:52773"

   volumes:

    - /external:/external

   environment:

    - ISCDATADIRECTORY=/external/ifconfig

   entrypoint: /iris-main --password-file /external/password.txt
```

Notice the similarity to the command line with a few additions. First the version of the docker-compose file is version ‘ 3 ’ . We have one service named ‘ iris ’ or another name of your choosing. The image, ports, volumes, and environment are self-explanatory. The last line “ entrypoint ” is the program that is run (including options, in the case the location of the password file).

Now if we type **docker-compose up -d**. The container is spun up the same as before. To stop the container type **docker-compose down**. More literature on docker-compose options are found on the Docker website. https://docs.docker.com/compose/

#Beginner #Best Practices #InterSystems IRIS

Source URL:https://community.intersystems.com/post/launching-iris-using-docker