

## Containers Workshop – InterSystems Global Summit 2023

### Introduction

Welcome to the Containers Workshop at InterSystems Global Summit 2023! This workshop contains two parts. In Part 1, you will learn how to interact with basic containers and use InterSystems IRIS® data platform in containers. In Part 2, you will build your own containerized InterSystems IRIS application for selling coffee makers that is based on a prebuilt sample app.

By the time you have finished the workshop, you will be able to:

- Create, manage, and remove Docker containers.
- Distinguish between running containers and container images.
- Create new container images.
- Persist data using bind mounts.
- Use Durable %SYS directories for persisting InterSystems IRIS data.
- Create container images from build processes.
- Use Docker Compose for multi-container deployments.
- Use container registries to share and retrieve containers.
- Update containers in place while preserving your application and data.

Throughout this workshop, you will complete exercises and watch accompanying presentations from InterSystems experts. Supporting materials will be linked in the provided slide deck in case you miss any of the presentations or wish to revisit the materials later.

Enjoy the Containers Workshop!



## Contents

<b>Containers Workshop – InterSystems Global Summit 2023</b> .....	1
Introduction .....	1
Setup — Install Docker and Pull an InterSystems IRIS Container Image.....	3
<b>PART 1</b> .....	5
Exercise 1: Perform Basic Container Operations.....	5
Exercise 2: Build a Container Image.....	8
Exercise 3: Persist Data with Bind Mounts .....	10
Exercise 4: Pull and Run InterSystems IRIS Containers .....	12
Exercise 5: Create a Container Image via Build Process.....	16
Exercise 6: Use Docker Compose for Multiple Containers .....	19
<b>PART 2</b> .....	21
Exercise 7: Build Your Own Containerized Application .....	21
BONUS: Make Code and Data Changes to Your Application .....	28
<b>Summary</b> .....	31

## Setup — Install Docker and Pull an InterSystems IRIS Container Image

In this workshop, you will use Docker on your own machine so that you are equipped to continue experimenting after the workshop. To do this, you will need to install and configure Docker and download the base InterSystems IRIS container image that you will use in these exercises.

Note that you should also have at least 10 GB of free disk space on your machine prior to setup.

This workshop assumes a basic awareness of InterSystems IRIS and its features. If you are not familiar with InterSystems IRIS, watch the [What is InterSystems IRIS? overview video](#) (4m).

### 0.1 — Install Docker

Docker is a mainstream container runtime engine which you will use during this workshop. To install Docker on your laptop, follow the instructions linked below for your operating system:

- [Install Docker Desktop on Mac](#) (select tab for Intel chip or Apple silicon chip)
- [Install Docker Desktop on Windows](#)
  - During Windows setup, it is recommended that you use WSL 2 instead of Hyper-V, if your machine supports it.
  - While installing Docker, you may need to update WSL using the following command:  
`wsl --update`
  - It is also recommended that you use PowerShell rather than Windows Command Prompt for this exercise.
- [Install Docker Desktop on Linux](#)

Once you have completed the installation process for your operating system, run the command below to test that Docker is installed and running:

```
docker run hello-world
```

You should see the output below if your Docker is running correctly:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
7050e35b49f5: Pull complete
Digest:
sha256:fffb13da98453e0f04d33a6eee5bb8e46ee50d08ebe17735fc0779d0349e889e9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

### 0.2 — Pull an InterSystems IRIS Community Edition image

Next, pull the base images of InterSystems IRIS Community Edition that will be used in this workshop. This will enable you to move more quickly through the exercise without downloading the images.

1. Run the three single-line commands below to download the InterSystems IRIS container images.

```
docker pull containers.intersystems.com/intersystems/iris-
community:2023.1.0.229.0
```

```
docker pull containers.intersystems.com/intersystems/iris-community:2023.2.0.198.0

docker pull
containers.intersystems.com/intersystems/webgateway:2023.1.0.229.0
```

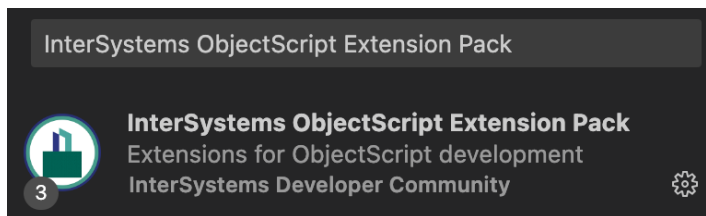
### 0.3 — Install Microsoft VS Code and InterSystems plugins

In the second half of this workshop, you will work with sample code to build a containerized application. To do this, you will need Microsoft Visual Studio Code and the InterSystems VS Code extension pack. Complete the following steps to set up these tools.

1. [Download VS Code](#) and follow the instructions for your operating system.
2. Once you have installed VS Code, open the Extensions tab on the left, represented by an icon of four squares.



3. Search for the InterSystems ObjectScript Extension Pack, and select the top result.



4. Click **Install** to install the extension pack.

This extension pack includes three different extensions: InterSystems ObjectScript, InterSystems Server Manager, and InterSystems Language Server.

### 0.4 — Windows only: Verify use of PowerShell

For Windows users, this exercise will assume that you are using PowerShell as your command line interface. Whenever the instructions tell you to open a new system-level terminal, or directly specify PowerShell, use PowerShell to run the exercise commands. If you cannot use PowerShell, contact an InterSystems helper to discuss an alternative.

## PART 1

### Exercise 1: Perform Basic Container Operations

In this exercise, you will become familiar with the basic commands and operations used with Docker containers. You will create and run a basic container, view your list of containers, and more.

To copy and paste commands and file entries throughout this workshop, you can use the accompanying `ContainersWorkshop-Commands.txt` file, which has plain text versions of every command without PDF formatting or spacing.

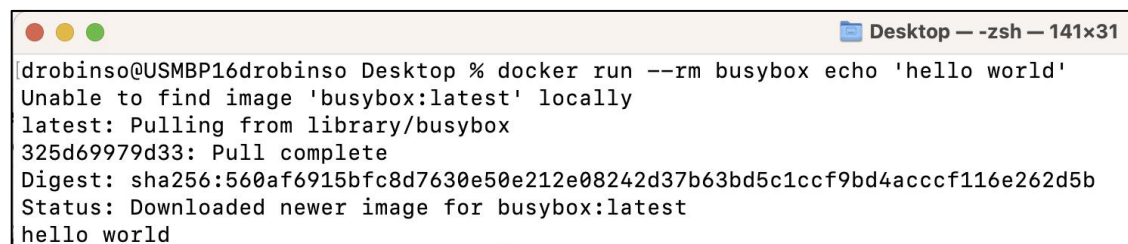
Throughout these exercises, you will see the leading `$` symbol, which indicates the start of the command line prompt on your system.

#### 1.1 — Create, run, and remove containers

1. Open a new Terminal or PowerShell window, depending on your system. Create and start a new temporary BusyBox container to ensure that Docker is installed correctly. If you do not have the BusyBox image locally already, Docker will download it first.

```
$ docker run --rm busybox echo 'hello world'
```

The `--rm` part of this command removes the container after the call is executed.

A screenshot of a terminal window titled "Desktop — zsh — 141x31". The prompt is "drobinso@USMBP16drobinso Desktop %". The command entered is "docker run --rm busybox echo 'hello world'". The output shows that Docker is pulling the 'busybox:latest' image from the library, with a pull complete message and a digest. Finally, it outputs "hello world".

```
drobinso@USMBP16drobinso Desktop % docker run --rm busybox echo 'hello world'
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
325d69979d33: Pull complete
Digest: sha256:560af6915bfc8d7630e50e212e08242d37b63bd5c1ccf9bd4acccf116e262d5b
Status: Downloaded newer image for busybox:latest
hello world
```

2. Create and start another BusyBox container. Include a `ping`, which will check if the container is running and accessible. Here, you are pinging `8.8.4.4`, which is a public Google DNS.

```
$ docker run busybox ping 8.8.4.4
```

3. Press **Ctrl+C** after three or four pings to stop the ping and exit the container.
4. List all your running containers using the `docker ps` command.

```
$ docker ps
```

There are currently no containers running, because you exited the BusyBox container. The `ps` command shows you the containers that are running on your machine. It includes useful information, such as the container name and ID and the image name. If you do not provide a name when initializing the container, Docker will generate a random name.

5. By adding the `--all` flag to `docker ps`, you can list all your containers, both running and stopped.

```
$ docker ps --all
```

Notice that the temporary BusyBox container from earlier is shown here, with a status of *Exited*. You can remove this container with the `docker rm` command.

drobinso@USMBP16drobinso	Desktop	%	docker ps --all				
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	
5b6aeff452cb	busybox	"ping 8.8.4.4"	29 seconds ago	Exited (0) 21 seconds ago		mystifying_feynman	

With most Docker commands, the inclusion of a `--help` flag will provide more information about the options for that command. Even running `docker --help` as its own command will provide a helpful overview of the commands you can run.

6. Browse the help results for the `ps` command help to see what options are available for modifying that command.

```
$ docker ps --help
```

7. Try some variants of the `ps` command, such as listing all containers by only their IDs.

```
$ docker ps --all --quiet
```

8. Remove the exited container, replacing `<CONTAINER ID>` with the ID of your exited container.  
**Note:** You can typically just provide the first few characters of the ID; Docker will recognize the ID when it has enough information to distinguish the ID as unique.

```
$ docker rm <CONTAINER ID>
```

9. Try viewing the help options for the `rm` command.

```
$ docker rm --help
```

By including a `--force` flag in an `rm` command, you could force the removal of a container that was still running.

You can use a command like `docker ps` as an argument for another command. For example, if you wanted to remove the most recent container that `docker ps` returned, you could include the command as the argument within an `rm` command.

10. Run a new BusyBox container, then run a command to remove containers returned by the `docker ps` argument.

```
$ docker run busybox echo 'hello world'
```

```
$ docker rm --force $(docker ps --latest --quiet)
```

The final command should return the ID of the removed containers.

```
drobinso@usmbp16drobinso ~ % docker run busybox echo 'hello world'
hello world
drobinso@usmbp16drobinso ~ % docker rm --force $(docker ps --latest --quiet)
8c93b50eea22
```

## 1.2 — Write files to a container

1. Run another BusyBox container, this time opening a shell on the container to interactively connect to it. You can do this by including the `-i` flag (for interactive mode) and the `-t` flag (to request a TTY connection). The trailing `sh` opens the bash shell.

```
$ docker run -it busybox sh
```

2. You should notice that your command prompt looks different now; the leading character is a `#` symbol. This indicates you are now in a command prompt within the container. Run the `ls` command to explore the current directory in the container's file system.

```
# ls -l
```

3. Create a new text file in the current directory and name it `test.txt`.

```
# echo 'Hello there...' > test.txt
```

4. List your files again to confirm that a text file has been created.

```
# ls -l
```

```
/ # echo 'Hello there...' > test.txt
/ # ls -l
total 44
drwxr-xr-x  2 root    root      12288 May 19 16:30 bin
drwxr-xr-x  5 root    root        360 May 22 19:46 dev
drwxr-xr-x  1 root    root      4096 May 22 19:45 etc
drwxr-xr-x  2 nobody nobody    4096 May 19 16:30 home
drwxr-xr-x  2 root    root      4096 May 19 16:30 lib
lrwxrwxrwx  1 root    root         3 May 19 16:30 lib64 -> lib
dr-xr-xr-x 236 root    root         0 May 22 19:45 proc
drwx----- 1 root    root      4096 May 22 19:46 root
dr-xr-xr-x 13 root    root         0 May 22 19:45 sys
-rw-r--r--  1 root    root       15 May 22 19:49 test.txt
drwxrwxrwt  2 root    root      4096 May 19 16:30 tmp
drwxr-xr-x  4 root    root      4096 May 19 16:30 usr
drwxr-xr-x  4 root    root      4096 May 19 16:30 var
/ #
```

5. Exit your container.

```
# exit
```

6. Run the same command you ran in Step 1.2.1 to start another container from the same image.

```
$ docker run -it busybox sh
```

7. Locate your `test.txt` file inside this new container using the `ls -l` command.

```
# ls -l
```

Your `test.txt` file is gone. Data stored directly in a container will be lost when the container is deleted. In later exercises, you will learn ways to persist data beyond the lifespan of an individual container.

8. Exit your container using the `exit` command.

## Exercise 2: Build a Container Image

Now that you have learned some of the basics around interacting with containers on your machine, let's build a new container image with some customizations to the base BusyBox container.

### 2.1 — Create a new container image

1. Run another BusyBox container, again initiating a command prompt on the container.

```
$ docker run -it busybox sh
```

2. Create an empty file called `myfile.test` on this container.

```
# touch myfile.test
```

3. List the files in the current directory to confirm `myfile.test` has been created.

```
# ls -l
```

```
/ #
/ # touch myfile.test
/ # ls -l
total 40
drwxr-xr-x  2 root    root      12288 May 19 16:30 bin
drwxr-xr-x  5 root    root        360 May 23 13:22 dev
drwxr-xr-x  1 root    root      4096 May 23 13:22 etc
drwxr-xr-x  2 nobody nobody    4096 May 19 16:30 home
drwxr-xr-x  2 root    root      4096 May 19 16:30 lib
lrwxrwxrwx  1 root    root         3 May 19 16:30 lib64 -> lib
-rw-r--r--  1 root    root         0 May 23 13:22 myfile.test
dr-xr-xr-x 240 root    root         0 May 23 13:22 proc
drwx----- 1 root    root      4096 May 23 13:22 root
dr-xr-xr-x 13 root    root         0 May 23 13:22 sys
drwxrwxrwt  2 root    root      4096 May 19 16:30 tmp
drwxr-xr-x  4 root    root      4096 May 19 16:30 usr
drwxr-xr-x  4 root    root      4096 May 19 16:30 var
/ #
```

4. Exit your container using the `exit` command. Then list all your containers, including the `--all` flag in your command to include both running and stopped containers.

**Note:** `-a` can be used as a shorthand for `--all`.

```
$ docker ps --all
```

5. Find the ID of your most recent container, which you created in Step 2.1.1. Use that ID as the `<CONTAINER ID>` in the `diff` command below. This command will show what has changed about a container relative to its image, so you can determine whether a container has changed enough from its base image to warrant the creation of a new image.

```
$ docker diff <CONTAINER ID>
```

The results of this command show you information about what has changed, depending on the letter that begins each line:

- Lines beginning with *A* show that a file or directory was added.



- Lines beginning with *C* show that a file or directory was changed.
- Lines beginning with *D* indicate that a file or directory was deleted.

```
drobinso@USMBP16drobinso Desktop % docker diff 00bcc
C /root
A /root/.ash_history
A /myfile.test
```

- When you created `myfile.test`, you wrote information to the container's read/write layer. Now, save that layer as a new image, which will serve as the basis for your next base image in developing your application.

With the command below, you will create a new container image that reflects the additions you made. Use the same ID as the previous step in place of `<CONTAINER ID>`.

```
$ docker commit <CONTAINER ID> myapp:1.0
```

This command names your new container image *myapp* and tags it *1.0*. Docker tags are simply labels or aliases given to images to help describe the image.

- Verify that this image has been created by listing all Docker images in your current repository.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myapp	1.0	a0238cd336be	10 seconds ago	4.86MB
busybox	latest	8135583d97fe	3 days ago	4.86MB
containers.intersystems.com/intersystems/iris-community	2023.1.0.229.0	0181aeb5d72f	5 weeks ago	2.03GB

- Finally, use what you have learned in the previous steps to run a new container using the `myapp:1.0` image. See if you can locate the `myfile.test` file, which is now included in all containers run from this version of the image.

Once you have found the file, exit your container. For an additional challenge and extra clean-up, retrieve all of the IDs for your existing BusyBox containers and remove those containers.

### Exercise 3: Persist Data with Bind Mounts

In Exercise 2, you created a new container image based on changes you made. But in most cases, you will want to persist data without continually making new container images so that your application is independent from your data and easy to modify and upgrade without losing the data your application is storing. In this exercise, you will use bind mounts to store data outside of the container while allowing it to be accessed in the container.

#### 3.1 — Use a bind mount to persist a container's data

1. Start a BusyBox container with a bind mount.

```
$ docker run -it --name mytest -v "${PWD}/mydata:/mydata" busybox
```

This command specifies the directory of the bind mount on the host file system and mounts that directory into the mydata directory of the container's file system.

You may get a pop-up indicating that Docker requires access to your current folder in order to create the mydata directory. Allow this access.

2. You should now be in a bash shell on this newly named mytest container. Run the following three commands in succession to enter your mydata directory, add a file, and exit the shell.

```
# cd mydata
# touch myfile.txt
# exit
```

3. Inspect your container to verify that the bind mount was created correctly.

```
$ docker inspect mytest
```

The Mounts section of this output shows that the directory you mounted in Step 3.1.1 is, in fact, being used.

```
{
  "Mounts": [
    {
      "Type": "bind",
      "Source": "/Users/drobinso/Desktop/mydata",
      "Destination": "/mydata",
      "Mode": "",
      "RW": true,
      "Propagation": "rprivate"
    }
  ]
}
```

4. To return only a filtered set of JSON results from the above inspect command, you can include additional criteria.

```
$ docker inspect --format='{{json .Mounts}}' mytest
```

Be sure to include the space between *json* and *.Mounts* in the above command.

5. Remove all of your containers. If you already had containers on your machine before this lab, you should carefully remove only the containers you have created in this exercise using their IDs.

```
$ docker rm --force $(docker ps -aq)
```

6. Verify your myfile.txt file still exists in the mydata folder on the host machine.

```
$ ls ./mydata
```

```
drobinso@USMBP16drobinso Desktop % ls ./mydata  
myfile.txt
```

## Exercise 4: Pull and Run InterSystems IRIS Containers

In this exercise, you will begin working with InterSystems IRIS containers to containerize applications built on InterSystems IRIS that may have traditionally run via local or server installations.

### 4.1 — Run InterSystems IRIS Community Edition from a script

1. Download or clone the [GitHub repository for this workshop](https://github.com/interSystems/Samples-Container-Workshop), either downloading it as a ZIP file or using the command below. Be sure to note the directory to which you have saved or cloned this repository, as the file paths in the following exercises will be relative to this directory.

```
$ git clone https://github.com/interSystems/Samples-Container-Workshop.git
```

2. Go to the location where you placed the repository, then navigate to the GS2023 directory that corresponds to your system (either GS2023-Linux or GS2023-Windows).

For Linux:

```
$ cd GS2023-Linux
```

For Windows:

```
$ cd GS2023-Windows
```

3. Run the script located in your appropriate GS2023 folder (run.bat for Windows, run.sh for Linux). This script includes the container image that you pulled during the setup process.

For Linux:

```
$ ./run.sh
```

For Windows:

```
$ ./run.bat
```

If you are prompted to enable permissions for your script to run, do so.

4. Locate your container ID from the output. You will need the first five digits of this ID.

```
drobinso@USMBP16drobinso GS2023-Linux % ./run.sh
8ea7200a18d875caa13c64faa296ce4614125a928ab983aa81226e98bab796de
```

### 4.2 — Persist Data on a New Container Using Durable %SYS

1. Enter the container you just created, using the first five digits of its ID in place of <CONTAINER ID> below. Notice that this time, instead of concluding your command with sh to enter a container shell, you are concluding with iris terminal iris to enter an InterSystems IRIS Terminal session on the container.

```
$ docker exec -it <CONTAINER ID> iris terminal iris
```

Your command prompt should now begin with USER>, indicating that you are in the USER namespace of your InterSystems IRIS container.

2. Run the command below to print your current directory.

```
USER>!pwd
```

```
USER>!pwd  
/ISC/iris.data.d/mgr/user
```

3. Next, set the value of a global called `^Kilroy`. Typically, you will not directly edit globals in InterSystems IRIS, but this is a quick way to demonstrate the addition of data within InterSystems IRIS.

```
USER>set ^Kilroy="I was here at " _$ZDT($H)
```

This command uses the ObjectScript function `$ZDT` and a special variable, `$H`. These are abbreviations in ObjectScript, which you can learn more about in [InterSystems documentation](#).

4. Write the `^Kilroy` global to observe its contents.

```
USER>write ^Kilroy
```

InterSystems containers have their time zone set to UTC, which is quite common for servers. As a result, the time entered in the `^Kilroy` global may be different from your machine's local time.

5. Keep this terminal window open (we will call this window Terminal 1) and open a second system-level terminal window (Terminal 2) on your machine. This should be either a Mac/Linux Terminal window or a Windows PowerShell window. Navigate to your working directory (the Linux or Windows GS2023 folder), then enter the durable directory using the command below.

```
$ cd durable
```

6. Run the `ls` command to view the contents of the durable directory. What do you notice?

```
$ ls
```

Within the durable folder, you can see a subdirectory called `iris.data.d`. This may look familiar—recall in Step 4.2.2, your working directory included `iris.data.d` in its path. This directory is the Durable %SYS directory, and it is usable both inside and outside of the container.

Optionally, you can open your `run.sh` (Linux) or `run.bat` (Windows) file to inspect it further. You will notice that this script utilizes a bind mount and the `$ISC_DATA_DIRECTORY` setting to configure this Durable %SYS directory.

#### 4.3 — Interact with your InterSystems IRIS container

1. Keeping both Terminal 1 and Terminal 2 windows open, use a web browser to access the Management Portal for your InterSystems IRIS container.

<http://localhost:80/csp/sys/UtilHome.csp>

2. Log in with username `_SYSTEM` and password `SYS`. You will be prompted to change the password. Change it to lowercase `sys`.

**Note:** By default, the Management Portal is accessible via web port 52773 for a local installation. When you are outside of the container and in your web browser, the port depends on what mapping was used when the container was created. In your `run.sh` script, the container's port 52773 was mapped to the local machine's port 80. Thus, accessing port 80 gets you to this container's Management Portal.

3. From Terminal 2—which is outside of your container—stop your container. Because the `run.sh` script included `--name` when you created the container, you can reference it by name, `GS2023`, instead of by ID.

```
$ docker stop GS2023
```

4. Run the `ps --all` command to see that your container is no longer present, even when showing stopped containers.

```
$ docker ps --all
```

The `run.sh` script included `--rm=true` when you created the container, which means the container will be removed when it exits. Keep in mind, though, that all of your files stored in the `Durable %SYS` directory (`iris.data.d`) will still exist.

5. View your Terminal 1 window. What has changed?

Previously, this terminal window was sitting at a `USER>` prompt, which indicated it was in an InterSystems IRIS Terminal session inside the container. Now, the prompt has reverted to a system level since the container has exited. If you refresh your Management Portal page, you will see that it is no longer reachable since the container has exited.

6. Close your Terminal 2 window and proceed using your Terminal 1 window. Run the `docker images` command to view your list of Docker images.

```
$ docker images
```

The InterSystems IRIS community edition image that your previous container was run from is still there.

drobinso@usmbp16drobinso GS2023-Linux % docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myapp	1.0	eb484e4e361d	11 minutes ago	4.86MB
busybox	latest	8135583d97fe	10 days ago	4.86MB
containers.intersystems.com/intersystems/webgateway	2023.1.0.229.0	1a690ab8d70c	6 weeks ago	300MB
containers.intersystems.com/intersystems/iris-community	2023.1.0.229.0	0181aeb5d72f	6 weeks ago	2.03GB

7. Ensure you are back in the GS2023 directory that matches your system, then run a new InterSystems IRIS container from the same image, once again using the run script that corresponds to your system.

For Linux:

```
$ ./run.sh
```

For Windows:

```
$ ./run.bat
```

You will see in your output that the container ID is different than the previous container ID; even though it uses the same image and may have the same data thanks to the Durable %SYS directory, it is still a unique container.

```
drobinso@USMBP16drobinso GS2023-Linux % ./run.sh
30469e5852f09025f45268e204e53a2754d56551634c472b348786cbc7f785a2
```

8. Enter an InterSystems IRIS Terminal session on this container.

```
$ docker exec -it GS2023 iris terminal iris
```

9. Write the contents of your ^Kilroy global to the console.

```
USER>write ^Kilroy
```

The global should now be present with the same contents that you stored on your previous container. Despite the container being removed and replaced by a new one, your data is safely persisted in the `iris.data.d` directory.

#### 4.4 — Review logs for your container

1. Exit the InterSystems IRIS Terminal session using the `halt` command.

```
USER>halt
```

2. Run the `docker logs` command to inspect your GS2023 container.

```
$ docker logs GS2023
```

These logs correspond to the `STDOUT` and `STDERR` for the container's primary process, or `PID 1`. The streams are used for collecting the output and error messages from a container's process.

3. Add a flag to the command to control what logs are displayed. Adding the `--tail` flag limits the display to a specified number of recent lines, in this case, five.

```
$ docker logs --tail 5 GS2023
```

4. To monitor the logs as they happen in real time, add the `--follow` flag. In this case, you will not see much activity happening if you are not actively using the InterSystems IRIS instance.

```
$ docker logs --follow GS2023
```

To stop following the logs, press **Ctrl+C**.

## Exercise 5: Create a Container Image via Build Process

Throughout the previous exercises in this workshop, you have created and interacted with containers via the command line. In some cases, you have been running `.sh` scripts with preconfigured command line entries. In this exercise, you will explore how to create containers utilizing configuration files and a build process. When scaling out your DevOps processes, you likely will want to have standardized files that can define how your images are built, rather than relying on command-line entries.

### 5.1 — Create an application and a Dockerfile

1. Create a new file in your GS2023 directory (either GS2023-Linux or GS2023-Windows) called `server.js`. You can do this via Notepad, Visual Studio Code, or the command line.
2. Enter the contents below into the `server.js` file.

```
var http = require('http');
var handleRequest = function(request, response) {
    response.writeHead(200);
    response.end("Hello World!");
}
var www = http.createServer(handleRequest);
www.listen(8080);
```

3. Save the contents of the `server.js` file.
4. Create a new file named `Dockerfile` (no file extension).

A Dockerfile outlines the steps Docker will take to create your container. This file includes steps like copying the `server.js` file you just created, specifying the base container image, and exposing the appropriate ports.

5. Enter the contents below into the `Dockerfile` file, then save it. Make sure your text editor does not automatically add a `.txt` extension; the Dockerfile should have no extension.

```
# Deriving our container from a prebuilt one
FROM node:16-slim
COPY server.js .
EXPOSE 8080
# Run the following default command when the container is run
CMD node server.js
```

In this Dockerfile, the container is being derived from a prebuilt one: `node:16-slim`. This is a publicly available, lightweight Node.js container image. Optionally, you can pull this image and inspect it with the following two commands:

```
$ docker pull node:16-slim
$ docker inspect node:16-slim
```



## 5.2 — Build a container image from your Dockerfile

1. Now that you have created a Dockerfile, use the `docker build` command to build an image according to the steps the file specifies. Recall that you can also run `docker build --help` to see more information about this command and its options.

```
$ docker build -t service:v1 .
```

This `docker build` command uses the Dockerfile located in the current directory, so ensure that you are still in either the `GS2023-Linux` or `GS2023-Windows` directory. The `-t` flag tags the container image, in this case with the tag `v1`.

Be aware that the period at the end of this command is required. This portion of the command specifies the directory in which to look for the Dockerfile to build the image. In this case, you are using the current directory, represented by the single period.

2. Review your images again to confirm that your new container image has been created.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
service	v1	57e56ce7639a	11 seconds ago	134MB
myapp	1.0	a0238cd336be	About an hour ago	4.86MB
busybox	latest	8135583d97fe	3 days ago	4.86MB
containers.intersystems.com/intersystems/iris-community	2023.1.0.229.0	0181aeb5d72f	5 weeks ago	2.03GB

3. Using the `docker history` command, look at the history of your new container image.

```
$ docker history service:v1
```

Inspecting the history, you will see that several previous layers of the image are labeled `<missing>`. This is because the published `node:16-slim` image does not include complete images from its entire history, but the history is still viewable for the image.

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
7b307d135b14	5 seconds ago	CMD ["/bin/sh" "-c" "node server.js"]	0B	buildkit.dockerfile.v0
<missing>	5 seconds ago	EXPOSE map[8080/tcp:{}]	0B	buildkit.dockerfile.v0
<missing>	5 seconds ago	COPY server.js . # buildkit	200B	buildkit.dockerfile.v0
<missing>	2 weeks ago	/bin/sh -c #(nop) CMD ["node"]	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ENTRYPOINT ["docker-entry...	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) COPY file:4d192565a7220e13...	388B	
<missing>	2 weeks ago	/bin/sh -c set -ex && savedAptMark="\$(apt-...	9.4MB	
<missing>	2 weeks ago	/bin/sh -c #(nop) ENV YARN_VERSION=1.22.19	0B	
<missing>	2 weeks ago	/bin/sh -c ARCH= && dpkgArch="\$(dpkg --print...	100MB	
<missing>	2 weeks ago	/bin/sh -c #(nop) ENV NODE_VERSION=16.20.0	0B	
<missing>	2 weeks ago	/bin/sh -c groupadd --gid 1000 node && use...	333kB	
<missing>	2 weeks ago	/bin/sh -c #(nop) CMD ["bash"]	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:00bc0eda6d2eb0f8a...	69.3MB	

4. Run a new container from this image, naming it `myservice` and mapping its port 8080 to your local machine's port 8080.

```
$ docker run -d --name myservice -p 8080:8080 service:v1
```

This command runs the container from the `service:v1` image in detached mode (`-d`), meaning the terminal prompt remains available after running the container.

The written output on the console is your full container ID.

```
drobinso@usmbp16drobinso GS2023-Linux % docker run -d --name myservice -p 8080:8080 service:v1
03a0dc9ae2cc6f4fd56f1f54de31bace2f58b90c9955f1de6dcbb60871c9030c
```

5. View the list of running containers to see that this container is running.

```
$ docker ps
```

6. To verify that the container is actually running the server .js application you created, run a curl command at the exposed port 8080. Alternatively, you can open the URL in the below command via a web browser.

```
$ curl http://127.0.0.1:8080
```

The result for either method should be a Hello World! message.

```
drobinso@usmbp16drobinso GS2023-Linux % curl http://127.0.0.1:8080
Hello World!%
```

7. Stop your container using the docker stop command. Once again, since you provided a name for your container, you can use its name instead of its ID.

```
$ docker stop myservice
```

8. Remove your container using the docker rm command.

```
$ docker rm myservice
```

## Exercise 6: Use Docker Compose for Multiple Containers

In Exercise 5, you used a Dockerfile to tell a build process how to create your container. Next, you will utilize Docker Compose to build a multi-container application.

### 6.1 — Run a pair of containers using a Docker Compose file

1. Create a new file in your working directory called *docker-compose.yml*. Like earlier, you can use either Notepad, Visual Studio Code, or the command line.
2. Add the following contents to the *docker-compose.yml* file. Refer to the provided plain text file, or the files in your solution folder, to ensure you have no extra indentation or line breaks.

```
services:
  irisone:
    image: containers.intersystems.com/intersystems/iris-
community:2023.1.0.229.0
    hostname: irisone
    ports:
      # 1972 is the SuperServer port by default
      - "9091:1972"
      # 52773 is the web server / management portal port
      - "9092:52773"

  iristwo:
    image: containers.intersystems.com/intersystems/iris-
community:2023.1.0.229.0
    hostname: iristwo
    ports:
      # 1972 is the SuperServer port by default
      - "9095:1972"
      # 52773 is the web server / management portal port
      - "9096:52773"
```

**Note:** This Docker Compose file specifies the containers you want to create, start, and manage. The *irisone* container will be accessible via ports 9091 and 9092 on the local machine, and the *iristwo* container will be accessible via ports 9095 and 9096 on the local machine.

If your file is showing any syntax errors, compare your indentation to the screenshot below. Each InterSystems IRIS container should have `image`, `hostname`, and `ports` indented at the same level beneath it.

```
🔥 docker-compose.yml
1  services:
2    irisone:
3      image: containers.intersystems.com/intersystems/iris-community:2023.1.0.229.0
4      hostname: irisone
5      ports:
6        # 1972 is the SuperServer port by default
7        - "9091:1972"
8        # 52773 is the web server / management portal port
9        - "9092:52773"
10   iristwo:
11     image: containers.intersystems.com/intersystems/iris-community:2023.1.0.229.0
12     hostname: iristwo
13     ports:
14       # 1972 is the SuperServer port by default
15       - "9095:1972"
16       # 52773 is the web server / management portal port
17       - "9096:52773"
```

3. Save the changes to your file. Then, run the `docker-compose up` command to start this set of containers.

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

4. View your list of running containers.

```
$ docker ps
```

5. View the logs for your Docker Compose set of containers.

```
$ docker-compose logs
```

The logging API that Docker Compose uses is similar to the regular Docker logging API that you used in Exercise 4. Many of the logging commands are the same between the two.

6. Run the command below to specify a similar set of logging criteria that you used earlier—in this case, getting the 10 most recent lines and following the results.

```
$ docker-compose logs --tail 10 --follow
```

When following a log, use the commands below to perform different actions:

- **Ctrl+S**: Pause live following
- **Ctrl+Q**: Resume live following.
- **Ctrl+C**: Exit follow mode

7. Exit follow mode when you are done viewing the logs. Use the `docker-compose down` command to stop and remove your set of containers.

```
$ docker-compose down
```

## PART 2

### Exercise 7: Build Your Own Containerized Application

Now that you have built a foundational knowledge of containers and InterSystems IRIS images, you can build an application that utilizes the skills you have developed. In Part 2 of this workshop, you will:

- Containerize a sample coffee shop application built on InterSystems IRIS.
- Run and explore the container, connecting via web ports.
- Publish your container to a shared registry.
- Make changes to your container and publish those changes.
- Change the base image of your containerized application without changing data or code.
- Use docker-compose to create multiple containers for both your application and a web gateway.
- Use a SQL client to add data to your containerized application.

#### 7.1 — Containerize the sample coffee shop application

1. Navigate to the `exercise7` directory in the GitHub repository you cloned at the beginning of the workshop. This directory will be the same for Linux and Windows users.

```
$ cd exercise7
```

2. Create a new file in this directory called *Dockerfile* via either Notepad, VS Code, or a command line.
3. Enter the following contents into the file. An explanation of each line is below.

```
FROM containers.intersystems.com/intersystems/iris-  
community:2023.1.0.229.0  
COPY app app  
COPY iris.script iris.script  
RUN iris start IRIS && iris session IRIS < iris.script && iris stop  
IRIS quietly  
ENV ISC_CPF_MERGE_FILE=$ISC_PACKAGE_INSTALLDIR/merge.cpf  
COPY merge.cpf $ISC_PACKAGE_INSTALLDIR
```

- Line 1: Defines that a new container will be based on the 2023.1 InterSystems IRIS Community Edition container.
  - Lines 2/3: Copy files from the current directory into the container. Line 2 copies the coffee maker shop application, and Line 3 copies a script that will load the code into InterSystems IRIS.
  - Line 4: Loads the coffee shop code into InterSystems IRIS.
  - Lines 5/6: Use CPF merge to tell InterSystems IRIS to load the CPF merge file when the container starts.
4. Save changes to the file.
  5. Build and tag your container based on the Docker file you just created.

```
$ docker build --tag coffee:v1 .
```

6. Run your container with both port mapping and a Durable %SYS directory configured.

```
$ docker run -d --publish 9999:52773 --publish 9998:1972 --volume  
"${PWD}/durable/docker:/durable" --env ISC_DATA_DIRECTORY=/durable/iris  
coffee:v1
```

In this command, you are telling Docker to run the container from the `coffee:v1` image.

- The `-d` flag runs this command in detached mode. Recall that this means the command prompt will become available when the command completes.
- The InterSystems IRIS private web server port (52773) will be mapped to port 9999 on the host machine.
- The InterSystems IRIS superserver port (1972) will be mapped to port 9998 on the host machine.
- The directory `durable/docker` will be mounted as `/durable` in the container.
- The `ISC_DATA_DIRECTORY` environment variable will tell the InterSystems IRIS container where to keep durable data.

The previous command may complete instantaneously, but it may take a minute for your InterSystems IRIS to become accessible on this new container.

7. Take a look at the web page that is accessible via the port you mapped in the run command. Navigate to the following URL:

<http://localhost:9999/csp/coffee/index.html>

This will bring you to a coffee maker app that shows a demo coffee maker market. It displays 10 coffee makers, their prices, and their manufacturers. There is also a search bar, and the option to create a new coffee maker.



## 7.2 — Work with a container registry

1. Log in to Docker Hub using the credentials provided below; if you are already logged in to Docker, you should first run the `docker logout` command so that you can log in to the shared account for this workshop.

If you do not want to log out of your Docker account, contact a helper and your Docker username can be added as a collaborator to this workshop's repository.

```
$ docker login
```

```
Username: gs23containers
```

```
Password: containers!!
```

2. View the shared Docker Hub container registry for this workshop, `gs23containers`, using this link: <https://hub.docker.com/r/gs23containers/coffee>
3. View the **Tags** tab and explore the tagged images already present in this repository. In the next step, you will choose a unique tag name for your image.
4. Tag and publish your container image to the `gs23containers` registry. Where the command below says `<YOURNAME>`, add your name or another unique identifier to tag your container in the shared registry.

```
$ docker tag coffee:v1 gs23containers/coffee:<YOURNAME>
```

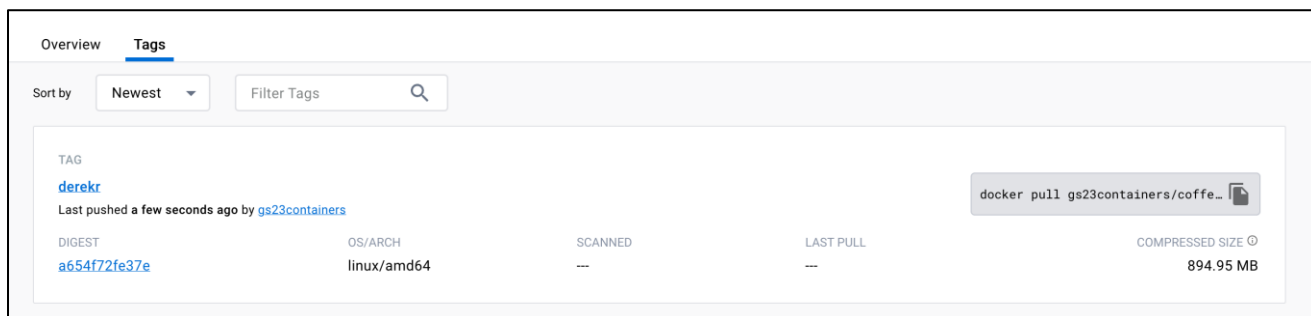
```
$ docker push gs23containers/coffee:<YOURNAME>
```

Recall that containers can have multiple tags, or names. In 7.1, you created a container simply named *coffee* and tagged it as version *v1*. In the first line above, you give that same container an additional name and tag so you can upload the container to the shared Docker Hub account for this workshop. With the second line, you upload, or push, the container to the registry.

Several layers of this container image should already exist in the repository, but one layer is modified for your unique image. The upload may take several minutes; if it is taking more than 3-4 minutes, consult a helper to determine whether or not to abandon the upload and continue.

5. When your push has completed, return to the shared registry in Docker Hub using the URL below and view the **Tags** tab. Can you find your container in the shared registry?

<https://hub.docker.com/r/gs23containers/coffee>



6. Pull your container image from the registry.

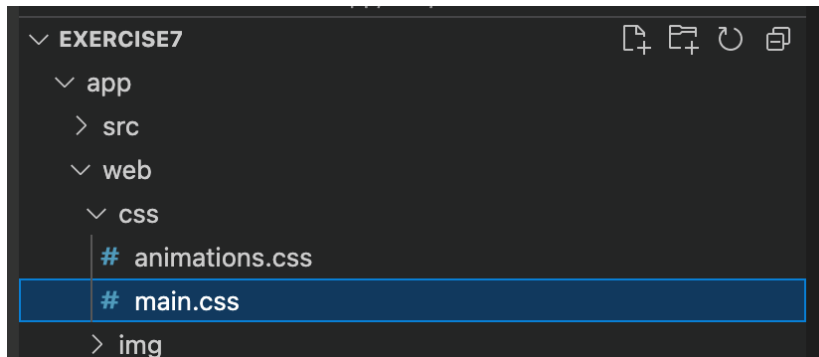
```
$ docker pull gs23containers/coffee:<YOURNAME>
```

This pull command executes very quickly because it does not actually pull any new data. That is because the container image on your machine perfectly matches the image stored in the registry, and Docker detects that there are no changes to be pulled.

```
drobinso@usmbp16drobinso ~ % docker pull gs23containers/coffee:derekr
derekr: Pulling from gs23containers/coffee
Digest: sha256:a654f72fe37e930f3ade26439efd63b2ab323c4dac4d8fe5b01b87c3e4bcb2ca
Status: Image is up to date for gs23containers/coffee:derekr
docker.io/gs23containers/coffee:derekr
```

### 7.3 — Make updates to your container

1. Introduce a new change to your containerized application. You can modify the `main.css` file, located in the `exercise7/app/web/css` folder of your repo, which defines the web styling of the coffee maker shop page.



If you are not sure what to change, try changing the background color of the coffee maker price tags. This is on line 47 of `main.css`. You can also change the background color of the coffee shop application, the title of the page, or another small visual change.

```
.price {
  background-color: #eba900 ;
  color: #fff;
  font-size: 18px;
```

2. Save the file you modified. Then build and run your container again, this time tagging it as version v2.

```
$ docker build --tag coffee:v2 .
```

3. Find the ID of your currently running `coffee:v1` container using the `docker ps` command. Then, stop that container using `docker stop` and the container's ID.

```
$ docker stop <CONTAINER ID>
```

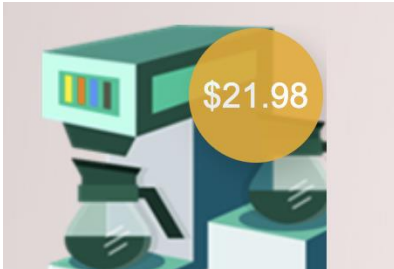


- Run the new version of your container using the command below.

```
$ docker run -d --publish 9999:52773 --publish 9998:1972 --volume  
"${PWD}/durable/docker:/durable" --env ISC_DATA_DIRECTORY=/durable/iris  
coffee:v2
```

- Refresh your application's front-end web application and either view the page in a new private browsing window or clear the cache on your browser. You can clear your cache in most browsers by holding **Shift** and clicking **Refresh**.

You should see that your changes have been introduced. For example, if you changed the color of the price tag, that should now be reflected.



- Publish this new version of the application to the container registry, tagging it with a v2 at the end of your name or identifier.

```
$ docker tag coffee:v2 gs23containers/coffee:<YOURNAME-v2>  
  
$ docker push gs23containers/coffee:<YOURNAME-v2>
```

Since you have introduced changes to this image, the push may take a few moments longer than the first time as changed layers of the container image are pushed to the registry.

```
drobinso@usmbp16drobinso GS2023-Linux % docker tag coffee:v2 gs23containers/coffee:derekr-v2  
drobinso@usmbp16drobinso GS2023-Linux % docker push gs23containers/coffee:derekr-v2  
The push refers to repository [docker.io/gs23containers/coffee]  
486e42d2ebec: Pushed  
1810c7ca29c4: Pushed  
5631b03e0d3f: Pushed  
961312bc7ddf: Pushed  
114f84796832: Layer already exists  
3fbdf4bc5f41: Layer already exists  
6b09c65411dc: Layer already exists  
53cb67f62f7d: Layer already exists  
a1dae3895e7e: Layer already exists  
a790f937a6ae: Layer already exists  
derekr-v2: digest: sha256:5302542f73986b49302b4ac4f33a8a557c413636a01361712795861011334a81 size: 2421
```

- Test out a newer version of InterSystems IRIS with your application, without changing the application or its data at all. Open your Dockerfile in VS Code to make a change to it. Change the FROM line of the Dockerfile to the following:

```
FROM containers.intersystems.com/intersystems/iris-  
community:2023.2.0.198.0
```

8. Rebuild your container from the updated Dockerfile.

```
$ docker build --tag coffee:v3 .
```

9. Stop your currently running coffee:v2 container, then run the command below to run your coffee:v3 container.

```
$ docker run -d --publish 9999:52773 --publish 9998:1972 --volume  
"${PWD}/durable/docker:/durable" --env ISC_DATA_DIRECTORY=/durable/iris  
coffee:v3
```

Notice that all your code and data remain unchanged, but your application is now running on an upgraded version of InterSystems IRIS. It can now take advantage of any new features that exist in the upgraded version of InterSystems IRIS.

**Note:** It is important to remember that you should never upgrade your InterSystems IRIS image directly on a production system without testing the upgrade first.

10. Once you are done using this container, clean up your Docker environment using the `docker stop` and `docker rm` commands you have learned in this workshop.

#### 7.4 — Use Docker Compose to run the application as multiple containers

1. Create a new Docker Compose file in your `exercise7` directory, using your preferred method.
2. Add the following contents to the `docker-compose.yml` file, then save the file.

```
version: '3.7'  
services:  
  # iris container  
  iris:  
    image: coffee:v3  
    init: true  
    container_name: iris  
    hostname: iris  
    ports:  
      - "8881:1972"  
    environment:  
      - ISC_DATA_DIRECTORY=/iris-shared/durable  
    volumes:  
      - type: bind  
        source: ./durable/compose  
        target: /iris-shared  
  
  # web gateway container  
  webgateway:  
    image:  
containers.intersystems.com/intersystems/webgateway:2023.1.0.229.0  
    init: true  
    container_name: webgateway  
    hostname: webgateway
```

```
ports:
- "8882:80"
- "8883:443"
environment:
- ISC_DATA_DIRECTORY=/webgateway-shared/durable
- ISC_CSP_CONF_FILE=/webgateway-shared/CSP.conf
- ISC_CSP_INI_FILE=/webgateway-shared/CSP.ini
volumes:
- type: bind
  source: ./webgateway
  target: /webgateway-shared
```

3. Run the `docker-compose up` command to deploy this set of containers. You are once again including the `-d` flag to ensure this command runs in detached mode.

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

4. Your containers should now be running. Navigate to the URL below to see the Management Portal for the web gateway container, accessible via port 8882 according to your mapping.

<http://localhost:8882/csp/sys/UtilHome.csp>

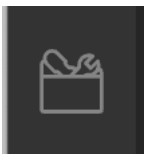
It may take a moment for your InterSystems IRIS instance to start and for the Management Portal to become available. By having this setup with a web gateway container alongside your application, you can avoid directly exposing your application to web access.

## BONUS: Make Code and Data Changes to Your Application

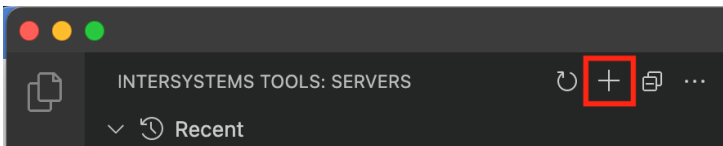
You have now taken a sample coffee maker shop application, originally built on a traditional instance of InterSystems IRIS, and containerized it. You have introduced some simple changes, tagged and published versions of the container, and used a shared container registry.

As a bonus activity, you will connect your VS Code workspace to the InterSystems IRIS instance running on your container and make some changes in ObjectScript. You will also connect DBeaver—a common SQL client—to the InterSystems IRIS database via JDBC to add data.

1. If you have not already been using it, confirm that you have installed VS Code and downloaded the InterSystems extensions. Refer to the instructions in the Setup section of this exercise guide if you have not yet done so. Once it is installed, you should see the InterSystems tools icon the left-hand navigation bar within VS Code.

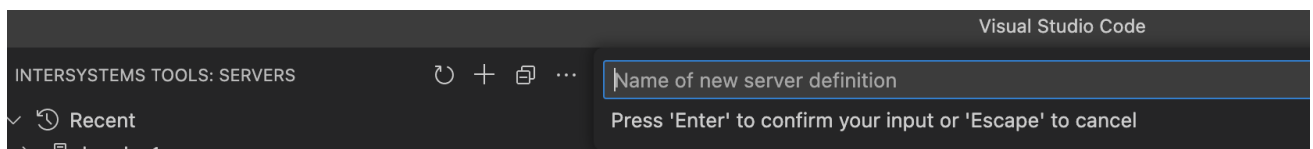


2. Click the InterSystems tools icon, and then click the + icon to add a new connection from VS Code to an InterSystems IRIS instance.



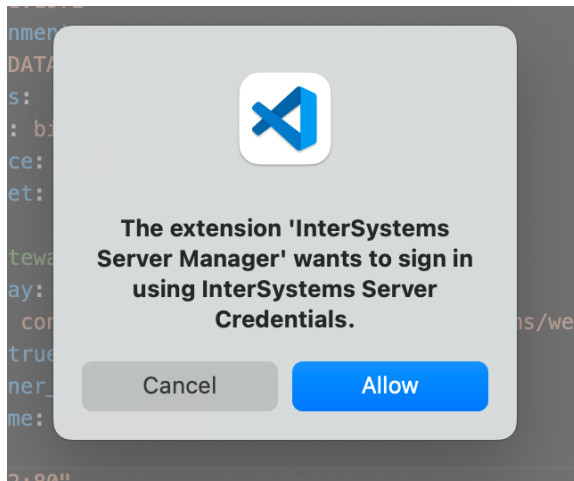
3. Enter the following information for your server:
  - **Name:** *summit*
  - **Description:** *Global Summit Container Workshop*
  - **Hostname:** *localhost*
  - **Port:** Enter the port for your InterSystems IRIS web server. If you're still running the docker-compose file, this will be *8882*.
  - **Username:** *\_SYSTEM*
  - **Protocol:** *http*

The prompts will appear in the top center of your VS Code window.

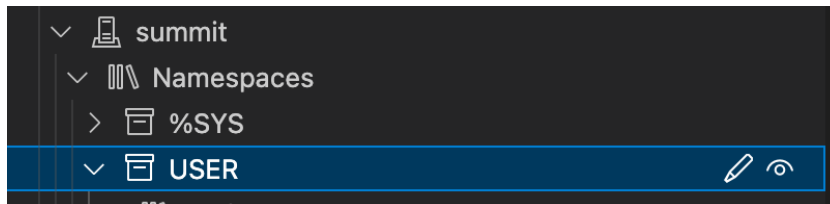


4. After entering this information, your server should be added to VS Code. Within the **All Servers** list in the left-hand menu, expand the **summit** server.

5. You will be prompted to allow InterSystems Server Manager to sign in with InterSystems Server Credentials. Click **Allow**.

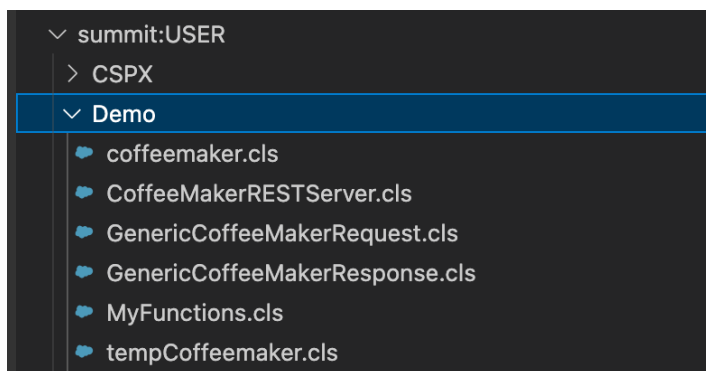


6. When prompted for the password, enter *SYS* and press **Enter**.
7. Expand the **summit** server and then expand the list of Namespaces. Next to the **USER** namespace, press the pencil icon to edit code in this namespace.



As you navigate through areas of VS Code, you may be prompted to select which user you want to use for the action. You can continue to select the *\_SYSTEM* user.

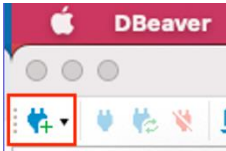
8. Explore the code in this namespace, including the Demo package that includes most of the class files for your coffee maker application. Make a code change somewhere that will alter the behavior of your app.



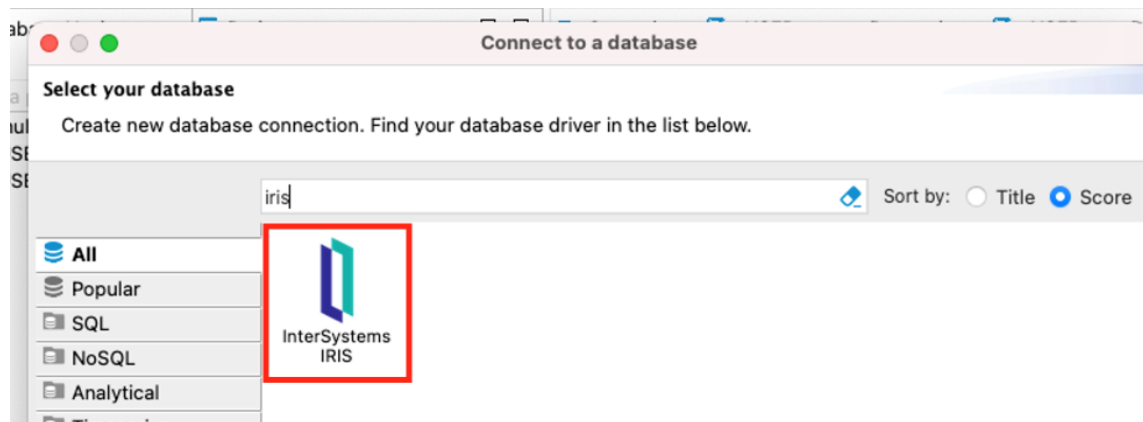
9. Add some data to your database via a SQL client. Begin by installing DBeaver from <https://dbeaver.io/> if you do not already have it on your machine.

**Note:** For an overview of the navigation pane, read the DBeaver online docs at <https://github.com/dbeaver/dbeaver/wiki/Application-Window-Overview>

10. Within DBeaver, create a new database connection by clicking the plug icon on the left-hand side of the toolbar. This brings up the “Connect to a database” dialog box.



11. Using the search bar, search for InterSystems IRIS. Ensure that the **All** tab is selected on the left side. Select InterSystems IRIS and click **Next**.



12. Enter the following connection properties:
  - **Host:** *localhost*
  - **Port:** This is the port for the InterSystems IRIS superserver. If you are still running the docker-compose example, enter *8881*.
  - **Database/Schema:** *USER*
  - **Username:** *\_SYSTEM*
  - **Password:** *SYS*
13. Click **Test Connection** to make sure the system is connected. Then click **Finish**. If you are prompted to do so, download any required drivers.
14. Right-click the connection and choose **SQL Editor > Open SQL Script**. This brings up an editor window. Enter the query below, then press the play arrow to execute it.

```
select * from Demo.coffeemaker
```

This command returns all data in the `Demo.coffeemaker` table.

15. Execute the insert statement below to add a new coffee maker to the database.

```
insert into Demo.coffeemaker (Name, Brand, CoffeemakerID, Color, Img, NumCups, Price) values ('Summit Coffee', 'GS23', 100, 'Teal', 'img/coffee10.png', 2, 39.95)
```

16. Return to your application's front end and explore it again to verify that the changes you made are reflected.
17. When you are done working with this application, bring down your Docker Compose environment using the docker-compose down command.

```
$ docker-compose down
```

18. Log out of the shared Docker credentials you used earlier, since they will not continue to work beyond this workshop.

```
$ docker logout
```

## Summary

Congratulations on finishing the Containers Workshop at InterSystems Global Summit 2023! In this workshop, you learned how to work with Docker containers and InterSystems IRIS Community Edition images. Containers provide a portable and efficient way to develop, deliver, and scale your applications for any environment. Whether you are building your application to be deployed in a public or private cloud, on a set of bare metal servers, or on your own machine, containers allow you to keep the same development approach and preserve a clean separation of your code and your data.

Using InterSystems IRIS in containers, you can continue to build the powerful applications that leverage the scalability, data model flexibility, and performance of InterSystems IRIS without sacrificing the efficiency and portability that containers provide.

And most importantly, you can now make the most of the following 2023 Global Summit sessions that show you even more of what you can do with container and cloud technologies!

Session	Presenters	Time & Location
Container Lifecycle: When is Your App Ready to Accept Work?	Bob Kuszewski Kerry Kirkham	Monday, 4:30PM – 4:50PM <i>Diplomat 3</i>
Embedded Source Control with Git & Containers	Pravin Barton	Monday, 4:30PM – 5:15PM <i>Diplomat 2</i>
Best Practices for InterSystems IRIS System Performance in the Cloud	Murray Oldfield Cliff Mason	Monday, 1:30PM – 2:15PM <i>Diplomat 1</i>
InterSystems Cloud Services: InterSystems IRIS SQL & IntegratedML	Iris Li Bhavya Kandimalla Steve LeBlanc	Monday, 3:30PM – 4:15PM <i>Diplomat 1</i>
FHIR in the Cloud	Pat Jamieson Bob Kuszewski	Tuesday, 2:30PM – 3:15PM <i>Regency 3</i>