Wifi: GSExpLab
Password: gs19boston
https://gs.learning.intersystems.com/event/gs19-container/

Containerizing Apps with InterSystems IRIS

Table of Contents

Introduction

Objective: In this experience lab, you will learn how to build an InterSystems IRIS-based application in a Docker container using the containerization capabilities in InterSystems IRIS.

Task: You will start with an InterSystems IRIS application (code provided) that has always been deployed in a traditional way. In this lab, you will containerize the app in order to leverage the lightweight, portable, and self-contained nature of Docker containers.

You will use the Docker build process, along with all its dependencies (Dockerfile, application dependencies) to create a final artefact that is portable, efficient, and deployable in cloud environments.

The URL below brings you to a video introducing this exercise and explaining the benefits of containerizing your applications. For the best viewing experience, enter the shortened URL below into your local browser or click the link in the PDF for this lab.

https://bcove.video/2ZiftD7

For more information, you can also view the video tutorial linked below on exporting source code from your environment via Atelier. You will not be required to export source code in this lab, but it is valuable for you to understand how. Again, enter the URL below into your local browser or click the link from the PDF for this lab.

https://bcove.video/2UMKSYF

Using This Notebook

The contents of this lab are runnable within this Jupyter Notebook on your provided VM. Below are a few key points to understand when using the notebook:

- Most of the runnable commands in the lab could also be executed in a terminal session on the VM itself. They are provided directly within the notebook for convenience.
    - Press **Enter** to run a block you have selected.
        - For each runnable module, [*] will display beside the module if it is still running. The module execution number will display once it has completed.
    - Press **Shift-Enter** or double-click to edit a block you have selected.
    - For additional help using the notebook, use the *Help* menu at the top of the screen.

Note: All commands that you run in this notebook are run in real time; they are not pre-recorded. For example, you can run the date command below to return the current date and time:

```
date
```

Get started by setting the current directory to gs19 and run the pwd command to ensure that you are in the gs19 directory:

```
cd ~/gs19
pwd
```

Running Docker Commands

By running docker container ls, you can see a list of running containers on your machine:

```
docker container ls
```

As expected, there are no running containers at the moment. Similarly, running docker volume ls shows you a list of volumes of which Docker is aware:

```
docker volume ls
```

Phase #1 — Build

In the first part of this exercise, you will use the InterSystems IRIS Community Edition, which is free for developers to use and does not require a license or paid subscription.

Run the docker pull command to pull the specified image of InterSystems IRIS from the Docker store. For your convenience, this image has already been pulled on your VM, so Docker will report that it is up to date.

```
docker pull store/intersystems/iris-community:2019.3.0.302.0
```

Run the docker images command to view a list of all Docker images you have on your machine. You can see the repository, tag, ID, creation date, and size of each image here.

```
docker images
```

Now, you will build a Docker image with your InterSystems IRIS application. In this phase, you will:

- Create an Application Installer Class for InterSystems IRIS.
- Create a Dockerfile for Docker, including API calls from %SYS.Container.
- Build your Docker image.

1. Create the Application Installer Class for InterSystems IRIS

Run the block below to create the application installer class. A done message will print when the code has finished running.

```
cat >AppInstaller.cls <<'EOF'
Class Util.AppInstaller
{
Parameter Namespace = "BC";

XData Install [ XMLNamespace = INSTALLER ]
{
<Manifest>

<Log Text="Creating namespace ${Namespace}" Level="0"/>
 <Namespace Name="${Namespace}" Create="yes" Code="${Namespace}-APP" Ensemble="" Data="USER">
  <Configuration>
   <Database Name="${Namespace}-APP" Dir="/opt/bc/db/app" Create="yes" MountRequired="true" Resource="
%DB_DEFAULT" PublicPermissions="RW" MountAtStartup="true"/>
  </Configuration>
  <CSPApplication  Url="/bc" Directory="/usr/irissys/mgr/web" AuthenticationMethods="64" IsNamespaceDefault="
false" Grant="%ALL" Recurse="1" />
  <Import File="/opt/bc/source/ui" Recurse="1" Flags="cuk" />
 </Namespace>
</Manifest>
}

/// This is a method generator whose code is generated by XGL.
/// Main setup method
ClassMethod RunManifest(ByRef pVars, pLogLevel As %Integer = 0, pInstaller As %Installer.Installer) As %Statu
s [ CodeMode = objectgenerator, Internal ]
{
    Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "Install")
}
/// Entry point
ClassMethod Run() As %Status
{
   try {
      write "START INSTALLER",!
      set vars("Namespace") = ..#Namespace
      set sc = ..RunManifest(.vars)
      write !,$System.Status.GetErrorText(sc),!
      if sc write !,"INSTALLER SUCCESS",!
      else  do $SYSTEM.Process.Terminate($JOB,1)
   } catch ex {
      set sc = ex.AsStatus()
      write $System.Status.GetErrorText(sc),!
      do $SYSTEM.Process.Terminate($JOB,1)
   }
   quit sc
}

}
EOF
echo done
```

2. Create the Dockerfile

Your Dockerfile contains the commands needed for Docker to automatically build your image. Later you will use this Dockerfile with a docker build command to create an image for your application. Once again, a done message will display when this module has finished running.

Note, specifically, that there is a method call from the SYS.Container API that is included in this Dockerfile:

```
do ##class(SYS.Container).QuiesceForBundling()
```

- This method calls all of the methods in this class that optimize InterSystems IRIS for being serialized safely into a container image.

The SYS.Container class is built into InterSystems IRIS and contains methods that help to optimize the containerization process of your applications.

```
cat >Dockerfile <<'EOF'
# Code installer example
#
# building from the InterSystems IRIS Community image

FROM store/intersystems/iris-community:2019.3.0.302.0

# we need to use Root user to set up environment
USER root

# copy all external dependencies
# you can use apt-get here
COPY figlet /usr/local/bin/figlet

# copy in application code and build class
COPY BC /opt/bc/source/
COPY AppInstaller.cls /opt/bc/install/

# change permissions to IRIS user
RUN chown -R ${ISC_PACKAGE_MGRUSER}:${ISC_PACKAGE_IRISGROUP} /opt/bc

# Change back to IRIS user
USER irisowner

# Compile the application code
RUN iris start iris && \
    printf 'zn "USER" \n \
    do $system.OBJ.Load("/opt/bc/install/AppInstaller.cls","c")\n \
    do ##class(Util.AppInstaller).Run()\n \
    zn "%%SYS"\n \
    do ##class(SYS.Container).QuiesceForBundling()\n \
    h\n' | irissession IRIS \
&& iris stop iris quietly
EOF
echo done
```

3. Build Your Container Image

As mentioned previously, when building your container image, Docker will use the Dockerfile that you created — and, via those steps, your Application Installer file — to automatically build your image.

This process may take a few moments. Look for an output message indicating the image is successfully tagged to know that it has completed.

```
docker build . --tag barkcare:stable --no-cache
```

By running docker images, you can now see the image you have just built in the barkcare repository.

```
docker images
```

Phase #2 — Run

Now that you have built a container image, you can run your container. Note that you are instantiating a running container from your image — not running the image itself. In this phase, you will:

- Run a container from the image you have built.
- Access your container, seeing both the BarkCare app and the InterSystems IRIS access points.
- Run multiple containers at once on the same host.

1. Run Your Container

With the docker run command below, you will create an instance of your container. This command provides a name, maps exposed ports, creates a volume, and sets environment variables.

```
docker run \
  --name test \
  --detach \
  --publish 7000:52773 \
  --volume testVolume:/external \
  --env ISC_DATA_DIRECTORY=/external/durable \
  barkcare:stable
```

Run docker container ls to once again see the list of all running containers on the machine. This time, there is an entry for the new container, named *test*.

```
docker container ls
```

Run the above command several times until you see InterSystems IRIS is started (its status will become healthy instead of health: starting). The initial start-up of these containers takes a few moments, because a durable %SYS environment is being created for persistent storage. Subsequent start-ups of the container will be much faster.

2. Access Your Container

Once your container is ready, you can access your containerized application via this web browser. It may take a few moments to start up. Click the link below to open your BarkCare application; if the page initially does not connect, wait a few seconds and try again.

http://127.0.0.1:7000/bc/ui.Bc.cls

Note that you have additional options for accessing your application's InterSystems IRIS instance:

- To access the InterSystems IRIS Management Portal, you can navigate to http://127.0.0.1:7000/csp/sys/UtilHome.csp. The default username is *SuperUser*, and the default password is *SYS*. You will be prompted to change these credentials upon your first login.
- After opening the Ubuntu Terminal by clicking the terminal icon at the top left corner of your VM screen, you can connect to an InterSystems IRIS Terminal session. To connect, enter the two commands below into the terminal. The first enters into a bash shell on the test container, and the second enters into an InterSystems IRIS Terminal session.
    - docker exec -it test bash
    - iris terminal iris

You will be prompted for a username (*SuperUser*) and password (*SYS* or your new password) after the second command. You will then be in an InterSystems IRIS Terminal session in the USER namespace.

3. Run Multiple Containers at Once

Just as easily as you have set up and run one instance of your BarkCare application, you can run additional instances by simply running another container from your base image. This functionality can be useful in a number of situations, such as when working with multiple versions of an application or when testing an application with several different runtime settings.

In the docker run command below, notice that you will use a different name, map a different port, and identify a different volume than you did for the first container.

```
docker run \
  --name prod \
  --detach \
  --publish 8000:52773 \
  --volume prodVolume:/external \
  --env ISC_DATA_DIRECTORY=/external/durable \
  barkcare:stable
```

You will access this new container the same way you accessed the previous one; the only difference is the port number. After giving it a moment to start up, visit this container's BarkCare application at the link below. You may be prompted to log in and change the password.

http://127.0.0.1:8000/bc/ui.Bc.cls

As for the additional options, they are similar to before:

- Access the InterSystems IRIS Management Portal at http://127.0.0.1:8000/csp/sys/UtilHome.csp
- Access the InterSystems IRIS Terminal by opening a new terminal session and entering the following two commands:
    - docker exec -it prod bash
    - iris terminal iris

You can take another look at your running containers with the docker container ls command:

```
docker container ls
```

Additionally, you can observe the list of volumes that Docker recognizes, which should now have a volume for each container you have created:

```
docker volume ls
```

Phase #3 — Publish

Up to this point, you have taken an application — one that that has previously been deployed in a traditional way — and containerized it. You have built a Docker image, run two containers from that image, and accessed two running instances of the containerized application.

However, these containers are still running locally on your VM, and the image from which you built the containers is also stored locally. In this phase, you will:

- Push your container image to Docker Hub.
- Learn how to pull an image from Docker Hub and run a container from that image on your own machine.

1. Push Your Container Image to Docker Hub

With the docker tag command below, you will tag your image in the gs19lab repository with your unique username.

(!) IMPORTANT: In the command below, enter a unique tag for yourself after the trailing colon. One common approach is to use your first initial followed by your last name; for example, John Smith might enter *JSMITH*.

```
docker tag barkcare:stable gs19lab/barkcare:
```

Then, with the docker push command, you will push your newly tagged image to Docker Hub, in the gs19lab repository. Add your unique username again here.

```
docker push gs19lab/barkcare:
```

2. Pull Your Image on Your Own Machine

Once your docker push command has completed, click the link below to view the gs19lab repository and find your image.

https://hub.docker.com/r/gs19lab/barkcare/tags

If you have Docker installed locally on your machine, run the following command in your local terminal.

(!) IMPORTANT: Replace *USERNAME* with the unique tag you used in the previous step.

- docker run -d --name gs19 -p 12345:52773 gs19lab/barkcare:USERNAME

This command will download a container image of your project, with all external dependencies included, and run it on your local machine. A few additional notes about this command:

- -d tells Docker to run the container in daemon mode.
- --name gives the container a name (*gs19*), so that you can more easily manage it later.
- -p publishes port 52773 on the container as your local port 12345.

Once you have run this command, you can navigate to the URL below on your own machine to see the BarkCare app in action — this time, running from a container on your own machine.

http://127.0.0.1:12345/bc/ui.Bc.cls

When you are done with the container, you can stop it by running a docker stop command.

- docker stop gs19

Phase #4 — Maintain

From a maintenance standpoint, it is important to be able to easily and seamlessly upgrade your environments when a new version of your application is ready. The runnable module below will make the following changes to your application:

- Change the version number from *25* to *26*.
- Change the color of the BarkCare header from black to red.

Feel free to make additional changes here, if you want to experiment further.

```
cat >BC/ui/Bc.cls <<'EOF'
Class ui.Bc Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
   &html<
   <h1 style="color:red;"> BarkCare </h1>
   <h2> Electronic Pet Record v26</h2>
   <h2> #($get(^InstanceName))# </h2>
   <pre>
                                        ;\
                                       |' \
                                       ; : ;
                          _            ; : ;
                         / `-.        /: : |
<a href="javascript:">Pet Search</a>              |  ,-.`-.      ,': : |
                        \  :  `. `.     ,'-. : |
<a href="javascript:">Resource Schedule</a>          \ ;   ;  `-.__,'   `-.|
                         \ ;   ;  ::: ,::'`:.  `.
<a href="javascript:">Pet Registration</a>           \ `-. :  `   :.   `.  \
                          \  \   ,   ;   ,:   (\
<a href="ui.PetOfTheDay.cls">Pet of the day</a>                \   :., :.    ,'o)): ` `-.
                           ,/,' ;' ,::"'`.`---'   `.  `-._
<a href="ui.Setup.cls">Setup</a>                    ,/  :  ; '"     `;'       ,--`.
                         ;/   :; ;          ,:'    (   ,:)
                           ,.,:.    ; ,:., ,-._ `.    \"""/
                           '::'     `:'` ,'(  \`._____.-'"'
                             ;,   ;  `.  `. `._`-.  \\
                             ;:.  ;:       `-._`-.\  \`.
                              '`:. :        |' `. `\  ) \
                                ` ;:       |    `--\__,'
                                  '`       '
                                       ,'
                                     ,-'
```

```
#($zv)#

  </pre>
  >
  quit $$$OK
}

}
EOF
echo done
```

With these changes made, you can build a new image for your application and tag it *barkcare:beta*. This may take a few moments to build.

```
docker build . --tag barkcare:beta --no-cache
```

Now, create a new container based on newly built image and data volumes of your **test** container. Note that you will not run this container just yet.

```
docker create \
  --name testnew \
  --publish 7000:52773 \
  --volumes-from test   \
  --env ISC_DATA_DIRECTORY=/external/durable \
  barkcare:beta
```

Now, you can upgrade your test environment. Instead of performing traditional upgrade procedures, you can simply replace your old container with the new one. All of your data remains in the same place, as it is stored in your persistent data volume. However, your application code, InterSystems IRIS database, and any external dependencies are replaced, since they are external to the data volume.

```
docker stop test

docker start testnew
```

After giving it a moment to start, test your new application to make sure it has been upgraded and shows *v26* and red header font.

http://127.0.0.1:7000/bc/ui.Bc.cls

Containers make it extremely easy to go in both directions with these changes. For example, if a new release needs to be downgraded, using a container removes uncertainty and allows you to revert changes easily. Simply stop the new container and start the old one to backtrack:

```
docker stop testnew

docker start test
```

Further Resources

Recommended Global Summit Sessions

- **The Value of Developing with Containers** (Joe Carroll)
  - Tuesday 2:30 – Fairfield/Exeter
- **InterSystems IRIS Containers for Developers** (Sean Klingensmith)
  - Tuesday 3:30 – Fairfield/Exeter
- **Durable Data Storage with Containers** (Mark Bolinsky)
  - Tuesday 4:30 – Fairfield/Exeter
- **Building Data-Driven Web Apps** (Sergei Shutov)
  - Wednesday 11:00 – Salon A/B
- **Introduction to Kubernetes** (Luca Ravazzolo)
  - Wednesday 11:00 – Arlington

Recordings of these sessions will be available on learning.intersystems.com.

Further Online Resources

- First Look: InterSystems Products in Docker Containers (exercise)
- How Are Containers Different From Virtual Machines? (video)
- Using Atelier with Your BarkCare Application (video)
- Docker Containers and InterSystems IRIS (video playlist)
- Docker for Windows and the InterSystems IRIS Data Platform (article)
- Using Package Manager with InterSystems IRIS in Docker Container (article)
- What is a Container? (article)
- Running InterSystems Products in Containers (documentation)
- Best Practices for Writing Dockerfiles (Docker documentation)

Remember to leave feedback about your experience in this lab in the Global Summit mobile app! Thanks for attending!