Running InterSystems Products in Containers

Version 2019.4
2020-01-28

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Running InterSystems Products in Containers
InterSystems IRIS Data Platform Version 2019.4 2020-01-28

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

Running InterSystems Products in Containers

This article explains the benefits of deploying software using Docker containers and provides the information you need to deploy InterSystems IRIS® and InterSystems IRIS-based applications in Docker containers, using the Docker images provided by InterSystems. The article covers the following topics:

For an introduction to this topic, including a brief hands-on experience, see First Look: InterSystems IRIS in Docker Containers.

You can get a Docker container image for InterSystems IRIS Community Edition, which comes with a free temporary license, from the Docker Store; see Downloading the InterSystems IRIS Docker Image for details.

1 Why Containers?

Containers package applications into platform-independent, fully portable runtime solutions, with all dependencies satisfied and isolated, and thereby bring the following benefits:

- Containers cleanly partition code and data, providing full separation of concerns and allowing applications to be easily deployed and upgraded.
- Containers are very efficient; an application within a container is packaged with only the elements needed to run it and make it accessible to the required connections, services, and interfaces, and the container runs as a single operating system process that requires no more resources than any other executable.
- Containers support clean movement of an application between environments — for example, from development to test and then to production — thereby reducing the conflicts typical of departments with different objectives building in separate environments. Developers can focus on the latest code and libraries, quality developers on testing and defect description, and operations engineers on the overall solution infrastructure including networking, high availability, data durability, and so on.
- Containers provide the agility, flexibility, and repeatability needed to revolutionize the way many organizations respond to business and technology needs. Containers clearly separate the application provisioning process, including the build phase, from the run process, supporting a DevOps approach and allowing an organization to adopt a uniform more agile delivery methodology and architecture (microservices).

These advantages make containers a natural building block for applications, promoting application delivery and deployment approaches that are simpler, faster, more repeatable, and more robust.

For an introduction to containers and container images from an InterSystems product manager, see What is a Container? and What is a Container Image? on InterSystems Developer Community.

Docker containers, specifically, are ubiquitous; they can be found in public and private clouds and are supported on virtual machines (VMs) and bare metal. Docker has penetrated to the extent that all major public cloud "infrastructure as a service" (IaaS) providers support specific container services for the benefit of organizations reducing system administration costs by using Docker containers and letting the cloud provider handle the infrastructure.

InterSystems has been supporting InterSystems IRIS in Docker containers for some time and is committed to enabling its customers to take advantage of this innovative technology.

For technical information and to learn about Docker technology step-by-step from the beginning, please see the Docker documentation site.

2 InterSystems IRIS in Docker Containers

Because a Docker container packages only the elements needed to run a containerized application and executes the application natively, it provides standard, well-understood application configuration, behavior, and access. If you are experienced with InterSystems IRIS running on Linux, it doesn't matter what physical, virtual, or cloud system and distribution your Linux-based InterSystems IRIS container is running on; you interact with it in the same way regardless, just as you would with traditional InterSystems IRIS instances running on different Linux systems.

The following describes different aspects of how InterSystems IRIS uses containers.

- *InterSystems-provided images* — A *container image* is the executable package, while a container is a runtime *instance* of an image. InterSystems provides Docker images containing a fully-installed instance of InterSystems IRIS, as well as other associated images, as described in Using InterSystems IRIS Docker Images. You can also use an InterSystems IRIS image from InterSystems as the basis for an image containing your InterSystems IRIS-based application; for more information, see Creating InterSystems IRIS Docker Images.
- *The iris-main program* — The *iris-main* program enables InterSystems IRIS and other products to satisfy the requirements of applications running in Docker containers. The *entrypoint application*, the main process started when a container is started, is required to block (that is, wait) until its work is complete, but the command starting InterSystems IRIS does not run as a blocking process. The **iris-main** program solves this by starting InterSystems IRIS and then continuing to run as the blocking entrypoint application. The program also offers a number of options to help tailor the behavior of InterSystems IRIS within a container. For more information about **iris-main**, see The iris-main Program.
- *The durable %SYS feature* — Because a containerized application is isolated from the host environment, it does not write persistent data; whatever it writes inside the container is lost when the container is removed and replaced by a new container. Therefore, an important aspect of containerized application deployment is arranging for data to be stored outside of the container and made available to other and future containers.

  The durable %SYS features enables persistent storage of instance-specific data — such as user definitions, audit records, and the log, journal, and WIJ files — when InterSystems IRIS is run in a container, allowing a single instance to run sequentially in multiple containers over time. For example, if you run an InterSystems IRIS container using durable %SYS, you can upgrade the instance by stopping the original container and running a new one that uses the instance-specific data created by the old one. For information about upgrading, see Upgrading InterSystems IRIS Containers; for detailed information on durable %SYS, see Durable %SYS for Persistent Instance Data.

InterSystems Cloud Manager (ICM) provides automated deployment of InterSystems IRIS containers and others on cloud infrastructure it provisions, as well as existing virtual and physical infrastructure. For more information about using ICM to deploy containerized InterSystems IRIS instances, see *First Look: InterSystems Cloud Manager* and the *InterSystems Cloud Manager Guide*.

3 Container Basics

This section covers the important basic elements of creating and using Docker containers.

3.1 Container Contents

In essence, a Docker container runs a single primary process, which can spawn child processes; anything that can be managed by a single blocking process (one that waits until its work is complete) can be packaged and run in a container.

A containerized application, while remaining wholly within the container, does not run fully on the operating system (OS) on which the container is running, nor does the container hold an entire operating system for the application to run on. Instead, an application in a Docker container runs natively on the kernel of the host system, while the container provides only the elements needed to run it and make it accessible to the required connections, services, and interfaces — a runtime environment (including file system), the code, libraries, environment variables, and configuration files.

Because it packages only these elements and executes the application natively, a Docker container is both very efficient (running as a discrete, manageable operating system process that takes no more memory than any other executable) and fully portable (remaining completely isolated from the host environment by default, accessing local files and ports only if configured to do so), while at the same time providing standard, well-understood application configuration, behavior, and access.

The isolation of the application from the host environment is a very important element of containerization, with many significant implications. Perhaps most important of these is the fact that unless specifically configured to do so, a containerized application does not write persistent data, because whatever it writes inside the container is lost when the container is removed and replaced by a new container. Because data persistence is usually a requirement for applications, arranging for data to be stored outside of the container and made available to other and future containers is an important aspect of containerized application deployment.

3.2 The Container Image

A *container image* is the executable package, while a container is a runtime *instance* of an image — that is, what the image becomes in memory when actually executed. In this sense an image and a container are like any other software that exists in executable form; the image is the executable and the container is the running software that results from executing the image.

A Docker image is defined in a Dockerfile, which begins with a base image providing the runtime environment for whatever is to be executed in the container. For example, InterSystems uses Ubuntu 18.04 LTS as a base for its InterSystems IRIS images, so the InterSystems IRIS instance in a container created from an InterSystems image is running in an Ubuntu 18.04 LTS environment. Next come specifications for everything needed to prepare for execution of the application — for example, copying or downloading files, setting environment variables, and installing the application. The final step is to define the launch of the application.

The image is created by issuing a **docker build** command specifying the Dockerfile's location. The resulting image is placed in the Docker image registry of the local host, from which it can be copied to other Docker image registries.

3.3 Running a Container

To execute a container image and create the container — that is, the image instance in memory and the kernel process that runs it — you must execute three separate Docker commands, as follows:

1. **docker pull** — Downloads the image from the repository.
2. **docker create** — Defines the container instance and its parameters.
3. **docker start** — Starts (launches) the container.

For convenience, however, the **docker run** command combines three separate Docker commands, which it executes in sequence, and is the typical means of creating and starting a container.

The **docker run** command has a number of options, and it is important to remember that the command that creates the container instance defines its characteristics for its operational lifetime; while a running container can be stopped and then restarted (not a typical practice in production environments), the aspects of its execution determined by the **docker run** command cannot be changed. For instance, a storage location can be mounted as a volume within the container with an option in the **docker run** command (for example, **--volume /home/storage:/storage3**), but the volumes mounted in this fashion in the command are fixed for that instantiation of the image; they cannot be modified or added to.

When a containerized application is modified — for example, it is upgraded, or components are added — the existing container is removed, and a new container is created and started by instantiating a different image with the **docker run** command. The new container itself has no association with the previous container, but if the command creating and starting it publishes the same ports, connects to the same network, and mounts the same external storage locations, it effectively replaces the previous container. (For information about upgrading InterSystems IRIS containers, see Upgrading InterSystems IRIS Containers.)

Important: InterSystems does not support mounting NFS locations as external volumes in InterSystems IRIS containers.

Note: As with other UNIX® and Linux commands, options to **docker** commands such as **docker run** can be specified in their long forms, in which case they are preceded by two hyphens, or their short form, preceded by one. In this document, the long forms are used throughout for clarity, for example **--volume** rather than **-v**.

4 Installing Docker

The Docker Engine consists of an open source containerization technology combined with a workflow for building and running containerized applications. Docker images from InterSystems comply with the OCI support specification, and are supported on Docker Enterprise Edition and Community Edition, version 18.03 and later. Docker EE only is supported for production environments.

To install the Docker engine on your servers, see Install Docker in the Docker documentation.

Docker supports a number of different storage drivers to manage images. To run InterSystems IRIS images, you may need to change the default driver, which depends on the host on which the Docker Engine is installed. For more information about supported storage drivers, see Docker Storage Driver.

5 Creating and Running InterSystems IRIS Docker Containers

This section describes what you need to do to run InterSystems IRIS containers using InterSystems images or images you have created, including the following topics:

- Using InterSystems IRIS Docker Images
- Creating InterSystems IRIS Docker Images
- The iris-main Program
- Durable %SYS for Persistent Instance Data
- Containerization Tools Provided with InterSystems IRIS
- Deploying Customized InterSystems IRIS Instances
- Running InterSystems IRIS Containers
- Upgrading InterSystems IRIS Containers

5.1 Using InterSystems IRIS Docker Images

InterSystems IRIS images provided by InterSystems are available worldwide from repositories, and can be downloaded using the **docker pull** command. (Pulling an image from a repository can also be done within a Dockerfile.)

The following sections cover several important issues concerning the use of InterSystems IRIS images provided by InterSystems, including:

- Docker Platforms and Versions Supported by InterSystems
- Downloading the InterSystems IRIS Docker Image
- License Keys for InterSystems IRIS Containers
- Security for InterSystems IRIS Containers
- Discovering Defaults in InterSystems Images

5.1.1 Docker Platforms and Versions Supported by InterSystems

InterSystems supports use of the InterSystems IRIS Docker images it provides on Linux platforms, and the instructions and procedures in this document are intended to be used on Linux. Rather than executing containers as native processes, as on Linux platforms, Docker for Windows creates a Linux VM running under Hyper-V, the Windows virtualizer, to host containers. These additional layers add complexity that prevents InterSystems from supporting Docker for Windows at this time.

We understand, however, that for testing and other specific purposes, you may want to run InterSystems IRIS-based containers from InterSystems under Docker for Windows. For information about the differences between Docker for Windows and Docker for Linux that InterSystems is aware of as they apply to working with InterSystems-provided container images, see Using InterSystems IRIS Containers with Docker for Windows on InterSystems Developer Community; for general information about using Docker for Windows, see Getting started with Docker for Windows in the Docker documentation.

Docker images from InterSystems comply with the OCI support specification, and are supported on Docker Enterprise Edition and Community Edition, version 18.03 and later. Docker EE only is supported for production environments.

Not all combinations of platform and Docker version are supported by Docker; for detailed information from Docker on compatibility, see the Compatibility Matrix and About Docker CE.

5.1.2 Downloading the InterSystems IRIS Docker Image

To make the InterSystems IRIS Docker image from InterSystems available for use, you can:

- Download an InterSystems IRIS Community Edition image from the Docker Store's InterSystems IRIS Data Platform page.

  InterSystems IRIS Community Edition comes with a free built-in 13-month license (and some functionality restrictions). For more information, see Deploying InterSystems IRIS Community Edition on Your Own System in *Getting Started with InterSystems IRIS Community Edition*.

- Download the archive from InterSystems and load the image.

  InterSystems IRIS images are distributed as Docker tar archive files, available in the InterSystems Worldwide Response Center (WRC) download area. Once you have downloaded the tar file, you can make it available on your system using the **docker load** command, as follows:

```
docker load -i iris-2019.4.0.633.0-docker.tar.gz
c6dba2103a94: Loading layer [==================================================>]  1.217GB/1.217GB
1b229d298c03: Loading layer [==================================================>]  1.479MB/1.479MB
Loaded image: docker.intersystems.com/intersystems/iris:2019.4.0.633.0
```

```
$ docker images
REPOSITORY                                    TAG             IMAGE ID       CREATED       SIZE
docker.intersystems.com/intersystems/iris     2019.4.0.633.0  e3cd4844771e   13 days ago   1.39GB
acme/iris                                     stable          e3cd4844771e   13 days ago   1.39GB
centos                                        7.3.1611        262f7381844c   2 weeks ago   192MB
hello-world                                   latest          05a3bd381fc2   7 months ago  1.84kB
```

  You can tag the image if you want a simpler name:

```
$ docker tag docker.intersystems.com/intersystems/iris:2019.4.0.633.0 acme/iris:stable
```

- Download the image from your organizations's Docker repository, if it has already been placed there.

  If your organization has a Docker repository and the InterSystems IRIS image from InterSystems is already loaded, obtain the location of the repository and the needed credentials, then log into the repository and list the images, as follows:

```
$ docker login docker.acme.com
Username: pmartinez@acme.com
Pasword: **********
$ $ docker images
REPOSITORY          TAG             IMAGE ID          CREATED       SIZE
acme/iris           stable          15627fb5cb76      3 days ago    1.39GB
centos              7.3.1611        262f7381844c      2 weeks ago   192MB
hello-world         latest          05a3bd381fc2      7 months ag   1.84kB
```

For simplicity, these instructions assume you are working with the image acme/iris:stable

5.1.3 License Keys for InterSystems IRIS Containers

Like any InterSystems IRIS instance, an instance running in a container requires a license key (typically called iris.key). For general information about InterSystems IRIS license keys, see the "Managing InterSystems IRIS Licensing" chapter of the *System Administration Guide*.

The InterSystems IRIS Community Edition image available from the Docker Store (as described in the previous section) comes with a special free temporary license. Generally, however, license keys are not, and cannot be, included in an InterSystems IRIS container image. Instead, you must stage a license key in a storage location accessible to the container, typically a mounted volume, and provide some mechanism for copying it into the container, where it can be activated for the InterSystems IRIS instance running there.

The **iris-main** program, which runs as the blocking entrypoint application of an InterSystems IRIS container, provides two options for handling the license key that can be used in a **docker start** or **docker run** command starting an InterSystems IRIS container, as follows:

- The **--key** option copies the license key from the location you specify to the mgr/ directory of the InterSystems IRIS instance, which means that it is automatically activated when the instance starts. The license key must not be located on the local file system inside the container; typically, it is on a storage location mounted as a volume by the container with the **--volume** option of the **docker run** command.

  This option is most useful when you are dealing with a single InterSystems IRIS container and a single license key. The syntax of the option is as follows:

  ```
  --key <key_path>
  ```

  where *key_path* is the path to the license key from within the container. For example, if you mount as /external an external storage location that includes a license key in a directory called license/, the **--key** option would look like this:

  ```
  --key /external/license/iris.key
  ```

- The **--license-config** option lets you do the following:
  - Optionally configure the InterSystems IRIS instance in the container as a license server (see the "Managing InterSystems IRIS Licensing" chapter of the *System Administration Guide*), which enables it to serve license keys contained in the staging location you specify to itself .iand to instances in other containers in the deployment. Both failover partners in a mirror can optionally be configured as the license server.
  - Specify the ID of the license to be served to the instance, by itself if configured as a license server or by another instance already configured as a license server.

  The syntax of the option is as follows:

  ```
  --license-config "<licenseID> <host1>,<port1>,[<directory1>] [<host2>,<port2>,<directory2>]
  ```

  where the arguments are as follows:

  - *licenseID*

    The value of the LicenseID field in the license key to be served to the instance in the current container. This field is found in the **[ConfigFile]** section, for example:

    ```
    [ConfigFile]
    LicenseID=2380451964
    FileType=InterSystems License Rev-A.1
    ```

  - *host1*,*port1*

    The hostname and port of the configured license server. If these arguments specify the instance in the container, that instance is configured as a license server; if not, theu specify another instance that is already configurad as a license server. The default InterSystems IRIS license server port is 4002.

  - [*directory1*]

    If the *host1*,*port1* arguments specify the instance in the current container to be configured as a license server, identifies the staging location in which the license keys it is to serve are located. This directory must not be on the local file system inside the container; typically, it is on a storage location mounted as a volume by the container with the **--volume** option of the **docker run** command. If the *host1*,*port1* arguments do not specify the instance in the current container as the license server, but instead identify another instance, this argument is ignored.

  - [*host2*,*port2*,*directory2*]

    If the *host1*,*port1* arguments specify the instance in the current container to be configured as a license server and that instance is a failover member in a mirror, these arguments specify the other failover instance to be configured

as a license server and identifis the staging area for the licenses it is to serve. The contents of *directory1* and *directory2* must be the same; the simplest way to effect this is to place the license directory on a storage volume mounted by both containers. If the instance specified by the *host1*,*port1* arguments is not a mirror member, these arguments are ignored.

See The iris-main Program for a list of **iris-main** options, and Running InterSystems IRIS Containers for examples of using the **--key** and **--license-config** options.

5.1.4 Security for InterSystems IRIS Containers

When working with InterSystems IRIS images from InterSystems it is important to understand their security-related characteristics, including:

- Ownership and directories
- Authentication and passwords

Ownership and Directories

The InterSystems IRIS instance in a container created from an InterSystems image is always named **IRIS** and is owned by **irisowner**, UID 51773. Some files are owned by and some processes run as **irisuser**, UID 52773. The working directory in the container (containing files such as iris-main.log) is /home/irisowner/, while the registry directory is /home/irisowner/irissys/ and the installation directory (containing the mgr/ subdirectory) is /usr/irissys/.

Note: If the **irisowner** and **irisuser** accounts are not defined in the /etc/passwd file on the system hosting the container, they are represented by their UIDs on that system, for example when you are listing a mounted external volume on the host file system:

```
$ ls -l /home/pmartinez/durable/irissys total 40K
drwxrwxr-x 4  51773   52773   29 Nov  8 12:42 csp/
drwxr-xr-x 3  root    root    21 Nov  8 12:42 dist/
drwxrwxr-x 4  51773   52773   30 Nov  8 12:42 httpd/
-rw-rw-r-- 1  51773   52773  10K Nov  8 12:42 iris.cpf
-rwxrw-r-- 1  52773   52773  10K Nov  8 12:42 iris.cpf_20191108*
-rwxrw-r-- 1  52773   52773  10K Nov  8 12:42 _LastGood_.cpf*
drwxrwxr-x 9  51773   52773 4.0K Nov  8 12:42 mgr/
```

The defaults for the InterSystems IRIS superserver port number and web server port number, respectively, were chosen for the UIDs of these accounts because they are easily recognizable to InterSystems IRIS users.

For more information on these installation-related topics, see InterSystems IRIS Installation in the "Installing on UNIX®, Linux, and macOS" chapter of the *Installation Guide*.

InterSystems provides tools that allow you to determine these configuration details for InterSystems IRIS-based images that you create, as described in Containerization Tools Provided with InterSystems IRIS.

Authentication and Passwords

OS-based authentication (see About Operating-System–Based Authentication in the "About InterSystems Security" chapter of the *Security Administration Guide*) is enabled for the InterSystems IRIS instance in a container created from an InterSystems image, and password authentication is disabled for the owner (**irisowner**).

InterSystems IRIS is installed with several predefined user accounts, including the **_SYSTEM** account (see Predefined User Accounts in the "Users" chapter of the *Security Administration Guide*), The default password for the predefined accounts is **SYS**. You can use predefined credentials (such as **_SYSTEM**/**SYS**) to log into an InterSystems IRIS instance deployed in a container, but all the accounts are configured to require a password change on first login. This is true whether you log in using the **iris terminal** command or the Management Portal. If relying on these password changes to secure the instance, be sure to log into all of the predefined accounts immediately.

InterSystems IRIS is distributed with an API call, **SYS.Container.ChangePassword()**, that changes the password of all of an instance's enabled user accounts that have at least one role to the contents of a user-provided file. (The option of specifying a read-only password file is provided for compatibility with Docker Secrets, Kubernetes Secrets, and similar technologies.) The change is made during the instance's first startup, before login is possible. For information about the SYS.Container API, see SYS.Container API and Image Build Script.

The **SYS.Container.ChangePassword()** method can be called using the **iris-main --password-file** option. This option is useful in scripts and other automation; when using it, bear in mind the risks of committing the password to a file for any significant length of time. For information about the **--password-file** option, see The iris-main Program.

InterSystems also provides a script, changePassword.sh, that calls **SYS.Container.ChangePassword()**. For information about using this script, see Password Change Script.

To avoid the expiration of passwords 90 days after an InterSystems IRIS image is built, which would occur using the default settings, a containerized instance is configured so that the passwords of the instance owner and the predefined accounts do not expire.

All of these configuration details can be determined by you for InterSystems IRIS-based images that you create using the tools described in Containerization Tools Provided with InterSystems IRIS.

5.1.5 Discovering Defaults in InterSystems Images

Default values in InterSystems containers are exposed through the standard label mechanism so that needed information can be discovered using the **docker inspect** command, as shown in the following example. Users familiar with InterSystems technology will recognize the typical default ports and other useful information. (For information about formatting the output of this command, see Format command and log output in the Docker documentation.)

```
$ docker inspect -f {{json .Config.Labels}} intersystems/iris:2019.4.0.633.0
  "Labels": {
       "com.intersystems.adhoc-info": "", "com.intersystems.platform-version": "2019.4.0.633.0",
       "com.intersystems.ports.default.arbiter": "2188",
       "com.intersystems.ports.default.license-server": "4002",
       "com.intersystems.ports.default.superserver": "51773",
       "com.intersystems.ports.default.webserver": "52773",
       "com.intersystems.ports.default.xdbc": "53773",
       "com.intersystems.product-name": "IRIS",
       "com.intersystems.product-platform": "dockerubuntux64",
       "com.intersystems.product-timestamp": "Wed Jan 16 2019 00:37:59 EST",
       "com.intersystems.product-timestamp.iso8601": "2019-08-16T05:37:59Z",
       "maintainer": "InterSystems Worldwide Response Center <support@intersystems.com>",
       "org.opencontainers.image.created": "2019-08-16T07:57:10Z",
       "org.opencontainers.image.documentation": "https://docs.intersystems.com/",
       "org.opencontainers.image.title": "intersystems/iris",
       "org.opencontainers.image.vendor": "InterSystems",
       "org.opencontainers.image.version": "2019.4.0.633.0"
       }
```

5.2 Creating InterSystems IRIS Docker Images

There is more than one approach to creating a Docker image for InterSystems IRIS. One approach is to design the Dockerfile (see The Container Image, and in the Docker documentation Best practices for writing Dockerfiles) to do the following in building the image:

- Specify the base image (InterSystems containers are based on Ubuntu 18.04 LTS.)
- Update the base and set environment variables as needed.
- Download the InterSystems IRIS installation kit.
- Install InterSystems IRIS.

Images officially supported by InterSystems contain an internally developed program called **iris-main** that is used as the entrypoint application to aid in handling InterSystems IRIS inside a container; this means copying the program into the Dockerfile and declaring it as the entrypoint. The **iris-main** program is described in The iris-main Program.

Given the complexities involved, experience with images from InterSystems is generally a prerequisite for building your own image as described in the preceding.

Alternatively, the most common and recommended approach to building an image that includes your InterSystems IRIS-based application along with InterSystems IRIS is to base it on an existing image from InterSystems, adding the dependencies relevant to your own application solution. This means starting with an image in which InterSystems IRIS is already installed with **iris-main** as the entrypoint; whatever you include in the Dockerfile is executed subsequent to this, which means you can start and issue commands to the InterSystems IRIS instance.

For a simple but instructive example of this approach, see https://github.com/intersystems/container-tools/2019.3/demo/fortune. which includes the following:

- An XML file containing ObjectScript code to run the Linux **fortune** command every 10 seconds
- A Dockerfile that does the following:
  1. Starts with an InterSystems IRIS image as base.
  2. Copies in the XML code file.
  3. Installs the **fortune** command in the container.
  4. Starts InterSystems IRIS and uses **iris terminal** (see Connecting to an InterSystems IRIS Instance in the "Using Multiple Instances of InterSystems IRIS" chapter of the *System Administration Guide*) to compile the code, then shuts down the instance.

By updating the **FROM** command in the Dockerfile as needed, then issuing the **docker build** command in a directory containing both the Dockerfile and the _ZSTART.xml file, you can create an image that, when run interactively with the command **docker run -it** *image*, runs the code in the InterSystems IRIS instance, which outputs the result of the **fortune** command every 10 seconds.

As described in Ownership and Directories and Required Environment Variables, the InterSystems IRIS instance is owned and must be started by the user specified by the $ISC_PACKAGE_MGRUSER variable; in an InterSystems image, this variable is set to **`irisowner`**. However, the **apt-get** update and **fortune** installation must be executed by the root user. The Dockerfile therefore starts as **`root`** and then switches to **`irisowner`** to copy _ZSTART.xml and start the instance.

Note: Updating the **apt-get** package cache before running it, as in the sample Dockerfile, is a best practice that helps avoid errors in building your image.

In exploring this approach, you can use the InterSystems IRIS Community Edition image described in Downloading the InterSystems IRIS Docker Image as a base image. Bear in mind, however, that it includes some functionality restrictions.

An important consideration when creating Docker images is image size. Larger images take longer to download and require more storage on the target machine. A good example of image size management involves the InterSystems IRIS journal files and write image journal (WIJ). Assuming that these files are relocated to persistent storage outside the container (where they should be), as described in Durable %SYS for Persistent Instance Data, you can reduce the size of an InterSystems IRIS or application image by deleting these files from the installed InterSystems IRIS instance within the container. The imageBuildSteps.sh script provided with InterSystems IRIS is used by Intersystems to address this and other issues when building InterSystems IRIS images, and can help you do so; see SYS.Container API and Image Build Script for details.

5.3 The iris-main Program

There are several requirements an application must satisfy in order to run in a Docker container. The **iris-main** program was developed by InterSystems to enable InterSystems IRIS and its other products to meet these requirements.

The main process started by the **docker run** command, called the *entrypoint*, is required to block (that is, wait) until its work is complete. In the case of a long-running *entrypoint application*, this process should block until it's been intentionally shut down.

InterSystems IRIS is typically started using the **iris start** command, which spawns a number of InterSystems IRIS processes and returns control to the command line. Because it does not run as a blocking process, **iris** is unsuitable for use as the Docker entrypoint application.

The **iris-main** program solves this problem by starting InterSystems IRIS and then continuing to run as the blocking entrypoint application. The program also gracefully shuts down InterSystems IRIS when the container is stopped, and has a number of useful options. To use it, add the **iris-main** binary to a Dockerfile and specify it as the entrypoint application, for example:

```
ADD host_path/iris-main /iris-main
ENTRYPOINT ["/iris-main"]
```

Docker imposes these additional requirements on the entrypoint application:

- Graceful shutdown with **docker stop**

  Docker expects the main container process to shut down in response to the **docker stop** command.

  The default behavior of **docker stop** is to send the SIGTERM signal to the entrypoint application, wait ten seconds, and then send the SIGKILL signal, which is not an acceptable way to stop a database under production workloads. Instead, **iris-main** intercepts Docker's signals and invokes the **iris stop** command.

- Graceful startup with **docker start**

  When a container is stopped by means other than the **docker stop** command, for example when the Docker daemon is restarted or the host is rebooted, the entrypoint application must carry out whatever tasks are required to bring the container back up to a stable running state in response to the **docker start** command. As of this writing, **iris-main** does not have any special handling for an InterSystems IRIS instance that was brought down ungracefully, and instead relies on existing InterSystems IRIS functionality; it does, however, execute all operations specified using the --before and --after options (see the table that follows).

- Logging to standard output for capture by **docker logs**

  Docker expects the entrypoint application to send output to the container's standard output so the **docker logs** command can display it. The **iris-main** program adheres to this by default, sending all IRIS log content to standard output. If you wish, you can instead direct the output of a different file in the container — for example, your application's log — to container output using the **-log** option, for example:

  ```
  docker run iris --log /myapp/logs/myapp.log
  ```

In addition to addressing the issues discussed in the foregoing, **iris-main** provides a number of options to help tailor the behavior of InterSystems IRIS within a container. The options provided by **iris-main** are shown in the list that follows; examples of their use are provided in Running InterSystems IRIS Containers.

Options for **iris-main** appear after the image name in a **docker run** command, while the Docker options appear before it. As with the **docker** command, the options have a long form in which two hyphens are used and a short form using only one.

Note: The **iris-main** program is configured to append its logging output to previous output, so that, when the InterSystems IRIS container is restarted, some record of how and why it shut down remains available.

Option	Description	Default
-i *instance,* --instance=*instance*	Sets the name of the InterSystems IRIS instance to start or stop. (The instance in a container distributed by InterSystems is always named **IRIS**.	IRIS
-d true|false, --down=true|false	Stops InterSystems IRIS (using **iris stop**) on container shutdown	true

Option	Description	Default
-u true|false, --up=true|false	Starts InterSystems IRIS (using **iris start**) on container startup	true
-s true|false, --nostu=true|false	Starts InterSystems IRIS in single-user access mode	false
-k *key_file*, --key=*key_file*	Copies the specified InterSystems IRIS license key (see License Keys for InterSystems IRIS Containers) to the mgr/ subdirectory of the install directory.	none
-L *license_config*, --license-config *license_config*	Configures the license server and specifies licenses to be served. For an explanation of this option's arguments, see License Keys for InterSystems IRIS Containers; for examples of its use, see Running InterSystems IRIS Containers.	none
-l *log_file,* --log=*log_file*	Specifies a log file to redirect to standard output for monitoring using the **docker logs** command.	none
-b *command,* --before *command*	Sets the executable to run (such as a shell script) before starting InterSystems IRIS	none
-a *command,* --after *command*	Sets the executable to run after starting InterSystems IRIS	none
-e *command,* --exit *command*	Sets the executable to run after stopping InterSystems IRIS	none
-c *command* --create=*command*	Execute a custom shell command before any other arguments are processed	
-t *command* --terminate=*command*	Execute a custom shell command after any other arguments are processed	
-p *password_file*, --password-file=*password_file*	Change the default password for the predefined InterSystems IRIS accounts to the string contained in the file, and then delete the file. **Important:** This option is useful in scripts and other automation; when using it, bear in mind the risks of committing the password to a file for any significant length of time. The first manual login to InterSystems IRIS after the container starts includes a mandatory default password change; for more information, see Security for InterSystems IRIS Containers.	
--version	Prints the **iris-main** version	N/A
-h, --help	Displays usage information and exits	N/A

5.4 Durable %SYS for Persistent Instance Data

This section describes the durable %SYS feature of InterSystems IRIS, which enables persistent storage of instance-specific data when InterSystems IRIS is run within a container, and explains how to use it.

5.4.1 Overview of InterSystems IRIS Durable %SYS

Separation of code and data is one of the primary advantages of containerization; a running container represents "pure code" that can work with any appropriate data source. However, because all applications and programs generate and maintain operating and historical data — such as configuration and language settings, user records, and log files — containerization typically must address the need to enable persistence of program-related data on durable data storage. In the case of an InterSystems IRIS container, a mechanism that accomplishes the following two things is required:

- Saving a variety of instance-specific data for use by the instance and by the instance in an upgraded container that replaces it, including the log, journal and WIJ files and the system databases that contain user definitions and other security information as well as audit records.
- Indicating where this data is stored, and where it can be found when running the upgraded image to create the upgraded container.

The durable %SYS feature does this by storing the needed data on an external file system, which is mounted as a volume within the container and identified in an environment variable specified when the container is started. While the InterSystems IRIS instance remains containerized, its instance-specific data exists outside the container, just like the databases in which application data is stored, persisting it across container and instance restarts and making it available for upgrading the instance. (For more information on upgrading, see Upgrading InterSystems IRIS Containers.)

Important: To maintain separation of code and data, InterSystems recommends creating all InterSystems IRIS databases on the external file system using durable %SYS or another mechanism. If you do create a database within the container for testing or development purposes by defining it in your Dockerfile, the database file (*db_name*.DAT) is created read-only. To enable InterSystems IRIS to mount it as read-write, you must open a shell inside the container, for example with **docker exec -it** *container_name* **bash**, and use the **touch** command on the database file before you start the IRIS instance, for example:

```
$ sudo docker exec -it try-iris bash
# touch /usr/databases/DBTEST.DAT
```

5.4.2 Contents of the Durable %SYS Directory

The durable %SYS directory, as created when a container is first started, contains a subset of the InterSystems IRIS install tree, including but not limited to:

- The configuration parameter file (CPF), which is named iris.cpf. Additional versions of the file (older versions and _LastGood_.cpf) are created as with any InterSystems IRIS instance. (*Configuration Parameter File Reference*)
- The /csp directory, containing the web gateway configuration and log files. (*InterSystems Web Gateway Configuration Guide*)
- The file /httpd/httpd.conf, the configuration file for the instance's private web server. ("Supported Web Servers" in the online *InterSystems Supported Platform* document for this release.
- The /mgr directory, containing the following:
  - The IRISSYS system database, comprising the IRIS.DAT and iris.lck files and the stream directory, and the iristemp, irisaudit, iris and user directories containing the IRISTEMP, IRISAUDIT, IRIS and USER system databases. (System-Supplied Databases and IRISSYS Database and Custom Items in the "Namespaces and Databases" chapter of the *Programming Orientation Guide*)

- The write image journaling file, IRIS.WIJ (which may be relocated to achieve file system separation). (The "Write Image Journaling and Recovery" chapter of the *Data Integrity Guide*)
- The /journal directory containing journal files (which may be relocated to achieve file system separation). (The "Journaling" chapter of the *Data Integrity Guide*; see also Separating File Systems for Containerized InterSystems IRIS in this document)
- The /temp directory for temporary files.
- Log files including messages.log, journal.log, and SystemMonitor.log. Additional logs may be present initially and some are created as needed, for example backup and mirror journal logs. (Monitoring InterSystems IRIS Logs in the "Monitoring InterSystems IRIS Using the Management Portal" chapter of the *Monitoring Guide*.)

  Note: Durable %SYS activity is logged in the messages.log file; if you have any problems in using this feature, examine this log for information that may help. For information about how to read this log from outside the container, see The iris-main Program.

- The InterSystems IRIS license key file, iris.key, either at container start if it is included in the InterSystems IRIS image or when a license is activated while the container is running. (Activating a License Key in the "Managing InterSystems IRIS Licensing" chapter of the *System Administration Guide*)
- Several InterSystems IRIS system files.

5.4.3 Locating the Durable %SYS Directory

When selecting the location in which this system-critical instance-specific information is to be stored, bear in mind the following considerations:

- The availability of appropriate backup and restore procedures ("Backup and Restore" chapter of the *Data Integrity Guide*)
- Any high availability mechanisms you have in place (*High Availability Guide*)
- Available storage space and room for expansion (Maintaining Local Databases in the "Managing InterSystems IRIS" chapter of the *System Administration Guide*)

There must be at least 200 MB of space available on the specified volume for the durable %SYS directory to initialize. For various reasons, however, including operational files such as journal records and the expansion of system databases, the amount of data in the directory can increase significantly.

5.4.4 Running an InterSystems IRIS Container with Durable %SYS

To use durable %SYS, include in the **docker run** command the following options:

```
--volume /<external_host>:/<durable_storage>
--env ISC_DATA_DIRECTORY=/<durable_storage>/<durable_dir>
```

where *external_host* is the host path to the durable storage location to be mounted by the container, *durable_storage* is the name for this location inside the container, and *durable_dir* is the name of the durable %SYS directory to be created in the location. For example:

```
docker run --detach
  --publish 52773:52773
  --volume /data/dur:/dur
  --env ISC_DATA_DIRECTORY=/dur/iconfig
  --name iris21 --init intersystems/iris:2019.4.0.633.0
```

In this example, the durable %SYS directory would be /data/dur/iconfig outside the container, and /dur/iconfig inside the container.

Important: When running InterSystems IRIS containers, always use the **--init** option to indicate that an init process should be used as PID 1 within the container, ensuring that the usual responsibilities of an init system are performed inside the container. For more information on this option, see the Specify an init process in the Docker documentation.

InterSystems strongly recommends using bind mounts, as illustrated in the preceding example, when mounting external volumes for InterSystems IRIS containers on production systems. However, under some circumstances, such as testing and creating demos or anything that you want to be portable to platforms other than Linux, it is preferable to use named volumes, because they eliminate problems related to directory paths, permissions, and so on. For detailed information about each method, see Manage data in Docker in the Docker documentation.

InterSystems does not support mounting NFS locations as external volumes in InterSystems IRIS containers.

Note: The **--publish** option publishes the InterSystems IRIS instance's web server port (52773 by default) to the host, so that the instance's management portal can be loaded into a browser on any host.

To avoid potential problems with the Docker TCP stack, you can replace the **--publish** option with the **--net host** option, which lets the container publish its default socket to the host network layer. The **--net host** option can be a simpler and faster choice when the InterSystems IRIS container you are running will be the only such on the host. The **--publish** option may be more secure, however, in that it gives you more control over which container ports are exposed on the host.

When you run an InterSystems IRIS container using these options, the following occurs:

- The specified external volume is mounted.
- The InterSystems IRIS installation directory inside the container is set to read only.
- If the durable %SYS directory specified by the **ISC_DATA_DIRECTORY** environment variable, iconfig/ in the preceding example, already exists and contains a /mgr subdirectory, all of the instance's internal pointers are reset to that directory and the instance uses the data it contains. If the InterSystems IRIS version of the data does not match the version of the instance, an upgrade is assumed and the data is upgraded to the instance's version as needed. (For information on upgrading, see Upgrading InterSystems IRIS Containers.)
- If the durable %SYS directory specified by **ISC_DATA_DIRECTORY** does not exist, or exists and is empty:
  - The specified durable %SYS directory is created.
  - The directories and files listed in Contents of the Durable %SYS Directory are copied from their installed locations to the durable %SYS directory (the originals remain in place).
  - All of the instance's internal pointers are reset to the durable %SYS directory and the instance uses the data it contains.

  If for any reason the process of copying the durable %SYS data and resetting internal pointers fails, the durable %SYS directory is marked as incomplete; if you try again with the same directory, the data in it is deleted before the durable %SYS process starts.
- If the durable %SYS directory specified by the **ISC_DATA_DIRECTORY** environment variable already exists and contains data (file or subdirectories) but does not contain a /mgr subdirectory, no data is copied; the container does not start, and the reason (data other than durable %SYS in the directory) is logged to standard output by the **iris-main** program, as described in The iris-main Program.

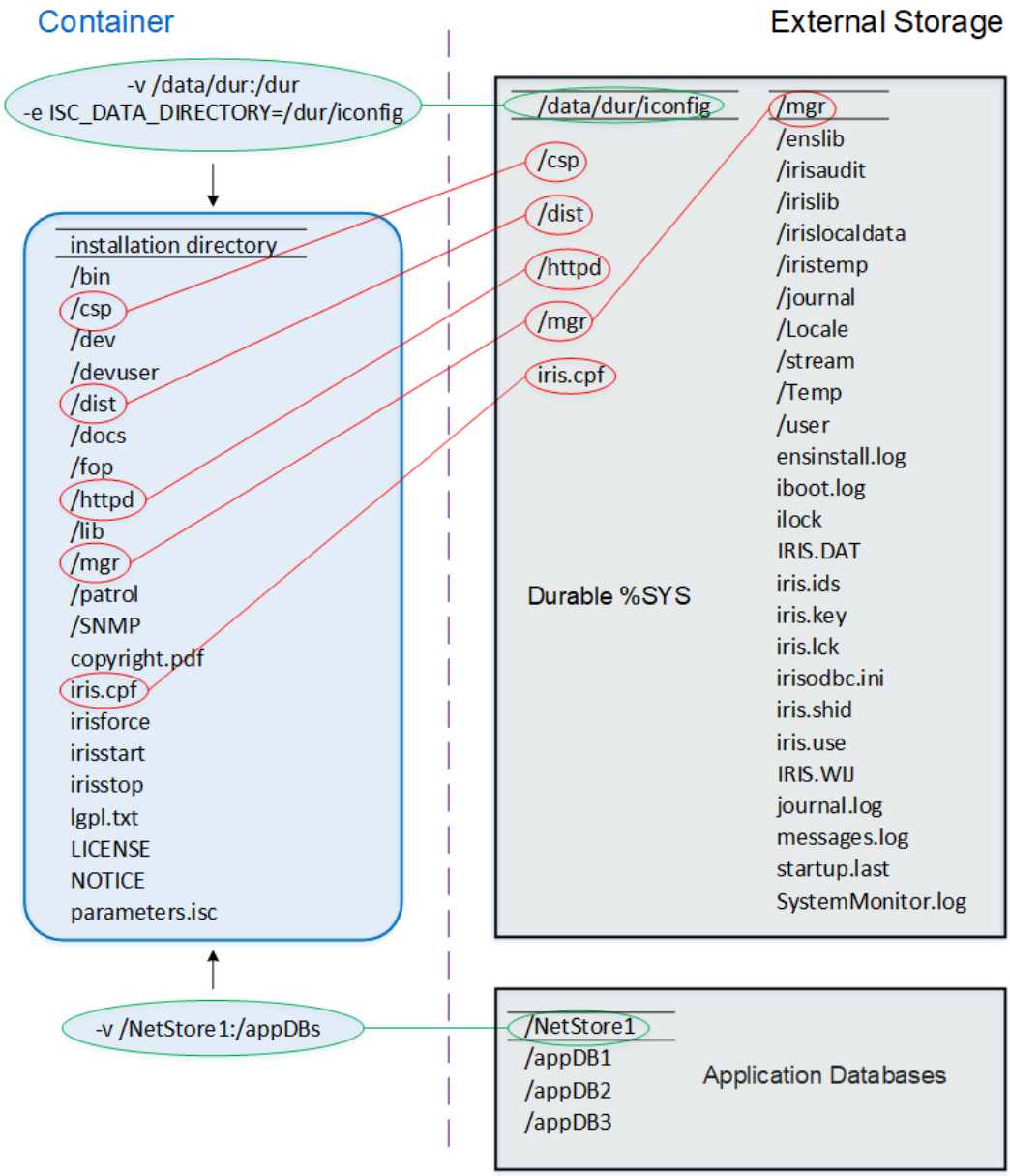
In the case of the example provided, the InterSystems IRIS instance running in container iris21 is configured to use the host path /data/dur/iconfig (which is the path /dur/iconfig inside the container) as the directory for persistent storage of all the files listed in Contents of the Durable %SYS Directory. If durable %SYS data does not already exist in the host directory

/data/dur/iconfig (container directory /dur/iconfig) it is copied there from the installation directory. Either way, the instance's internal pointers are set to container directory /dur/iconfig (host directory /data/dur/iconfig).

See Running InterSystems IRIS Containers for examples of launching an InterSystems IRIS container with durable %SYS.

The following illustration shows the relationship between the installation directory of a newly installed InterSystems IRIS container and the external durable %SYS directory, with external application databases also depicted.

Figure 1: InterSystems IRIS Installation Directory and Durable %SYS



5.4.5 Identifying the Durable %SYS Directory Location

When you want to manually verify the location of the durable %SYS directory or pass this location programmatically, you have three options, as follows:

- Open a shell inside the container, for example with **docker exec -it** *container_name* **bash**, and do either of the following:

```
echo $ISC_DATA_DIRECTORY
iris list
```

Note: For detail information on the **iris** command, see Controlling InterSystems IRIS Instances.

- Within InterSystems IRIS, call **$SYSTEM.Util.InstallDirectory()** or **$SYSTEM.Util.GetEnviron(ISC_DATA_DIRECTORY)**.

5.4.6 Ensuring that Durable %SYS is Specified and Mounted

When a container is run with the **ISC_DATA_DIRECTORY** environment variable, pointers are set to the durable %SYS files only if the specified volume is successfully mounted.

- If **ISC_DATA_DIRECTORY** is specified but the needed *--volume /external_host:/durable_storage* option is omitted from the docker run command, the instance fails to start and an error message is generated.
- If the **--volume** option is included but Docker cannot successfully mount the specified volume, it creates the external storage directory and the volume within the container; in this case, the instance data is copied to the durable %SYS directory, as described for "If the durable %SYS directory specified by **ISC_DATA_DIRECTORY** does not exist" in Running an InterSystems IRIS Container with Durable %SYS.

If **ISC_DATA_DIRECTORY** is not specified, the InterSystems IRIS instance uses the instance-specific data within the container, and therefore operates as a new instance.

To use durable %SYS, you must therefore ensure that all methods by which your InterSystems IRIS containers are run incorporate these two options.

5.4.7 Separating File Systems for Containerized InterSystems IRIS

In the interests of performance and recoverability, InterSystems recommends that you locate the primary and secondary journal directories of each InterSystems IRIS instance on two separate file systems, which should also be separate from those hosting InterSystems IRIS executables, system databases and the IRIS.WIJ file, with the latter optionally on a fourth file system. Following InterSystems IRIS installation, however, the primary and secondary journal directories are set to the same path, *install-dir*/mgr/journal, and thus may both be set to /mgr/journal in the durable %SYS directory when durable %SYS is in use.

After the container is started, you can reconfigure the external locations of the primary and secondary directories using the Management Portal or by editing the CPF (iris.cpf), as long as the volumes you relocate them to are always specified when running a new image to upgrade the InterSystems IRIS instance. You can also configure separate file systems using a CPF merge file, as described in Deploying Customized InterSystems IRIS Instances.

Note: When the durable %SYS directory is in use, the IRIS.WIJ file and some system databases are already separated from the InterSystems IRIS executables, which are inside the container. Under some circumstances, colocating the IRIS.WIJ file with your application databases instead may improve performance.

See File System Recommendations in the "File System and Storage Configuration Recommendations" chapter of the Installation Guide for more information about separation of file systems for InterSystems IRIS.

5.4.8 Containerization Tools Provided with InterSystems IRIS

InterSystems provides several containerization tools to aid you in creating your own InterSystems IRIS-based containers. This sections discusses the following topics:

- Required environment variables
- SYS.Container API and image build script
- Password change script

Required Environment Variables

There are a number of installation parameters available for use in configuring unattended installation of InterSystems IRIS instances on UNIX and Linux; their use is described and they are listed in Unattended InterSystems IRIS Installation in the "Installing InterSystems IRIS on UNIX®, Linux, and macOS" chapter of the *Installation Guide*. If you install InterSystems IRIS instance from a kit in your Dockerfile instead of using an InterSystems image as a base, as described in Creating InterSystems IRIS Docker Images, the installation parameters that are required as environment variables in the container runtime environment must also be built into the image; without them, container creation from the image will fail. These variables are included in all images from InterSystems and are shown, with the values set by InterSystems, in the following table:

Table 1: Installation Parameters Required as Environment Variables for Containerization

Parameter/Variable	Description	InterSystems Value
ISC_PACKAGE_INSTANCENAME	Name of the instance to be installed.	IRIS
ISC_PACKAGE_INSTALLDIR	Directory in which the instance will be installed.	/usr/irissys
ISC_PACKAGE_IRISUSER	Effective user for the InterSystems IRIS superserver.	irisuser
ISC_PACKAGE_IRISGROUP	Effective user for InterSystems IRIS processes.	irisuser
ISC_PACKAGE_MGRUSER	Username of the installation owner.	irisowner
ISC_PACKAGE_MGRGROUP	Group that has permission to start and stop the instance.	irisowner

Optionally, you can set the IRISSYS variable to specify the registry directory; InterSystems sets it to /home/irisowner/irissys in all images. If you do not include this environment variable, the registry directory is /usr/local/etc/irissys

Note: The environment variables discussed here are used to specify the configuration details described in Ownership and Directories.

SYS.Container API and Image Build Script

In building its InterSystems IRIS images, InterSystems uses the SYS.Container API to bring the installed InterSystems IRIS instance into a state in which it can safely be serialized into a container image. The class contains several methods that can be used individually, but one of these, **SYS.Container.QuiesceForBundling()**, calls all of the needed methods in a single operation, and is used by InterSystems in creating its images. Using this approach is the recommended best practice, because error-checking across the Linux shell/ObjectScript boundary is difficult and involves the risk of silent errors from IRIS; the fewer calls you make, the lower this risk is.

The SYS.Container code is included and fully visible in any InterSystems IRIS instance installed on Linux platforms; see the class reference for documentation. The methods include the following:

- **SYS.Container.QuiesceForBundling()**

  Calls all of the ObjectScript code necessary to get InterSystems IRIS into a state in which it can safely be serialized into a container image.

- **SYS.Container.ChangePassword()**

  Changes the password of all enabled user accounts with at least one role; called by the password change script.

- **SYS.Container.ChangeGatewayMgrPassword()**

  Changes the Web Gateway management password (see Web Gateway Management Pages in the "Web Gateway Operation and Configuration" chapter of the *Web Gate Configuration Guide*).

- **SYS.Container.ForcePasswordChange()**

  Sets **Change password on next login** on user enabled accounts with at least one role (see Properties of Users in the "Users" chapter of the *Security Administration Guide*).

- **SYS.Container.KillPassword()**

  Disables password-based login for a specified user; other forms of authentication (see Authentication: Establishing Identity in the "About InterSystems Security" chapter of the *Security Administration Guide*) remain enabled.

- **SYS.Container.EnableOSAuthentication()**

  Enables OS-based authentication for the instance (see About Operating-System–Based Authentication in the "About InterSystems Security" chapter).

- **SYS.Container.SetNeverExpires()**

  Sets **Account Never Expires** for the specified user account; without this, user accounts will expire in images that are more than 90 days old (see Properties of Users in the "Users" chapter of the *Security Administration Guide*).

- **SYS.Container.PreventFailoverMessage()**

  Prevents journal rollover messages from the instance in a newly started container.

- **SYS.Container.PreventJournalRolloverMessage()**

  Prevents the instance from posting a warning because the name of the host it is running on is not the same as the hostname stored from the last time it was running.

- **SYS.Container.SetMonitorStateOK()**

  Clears level 1 and level 2 alerts from the System Monitor, generating an error if a level 3 is present (see the "Using System Monitor" chapter of the *Monitoring Guide).*

Note: The methods listed here can be used to specify the configuration details described in Authentication and Passwords.

To assist you in using the SYS.Container API, InterSystems also includes provides the script used to build its images, imageBuildSteps.sh, which can be found in the following locations:

- In dev/Container/ under the InterSystems IRIS installation directory on Linux platforms.
- At https://github.com/intersystems/container-tools/tree/master/2019.3/official/iris.

The imageBuildSteps.sh can be called in the Dockerfile following the installation of the instance, for example:

```
RUN irisinstall_silent && imageBuildSteps.sh
```

The script includes the following steps:

1. Verifies that the required environment variables are set.
2. Brings the instance up in single-user mode, which bypasses all forms of authentication and allows only the instance owner to log in.
3. Configures **SYS.Container.ErrorHandler()** to terminate on error and display error messages (the default settings).
4. Calls **SYS.Container.QuiesceForBundling()**.
5. Shuts down the instance.

6. Removes the WIJ and journal files to minimize the size of the image.
7. If the IRISSYS variable is set (see Required Environment Variables), configures the ISCAgent accordingly.

Password Change Script

The change password script, changePassword.sh, is provided for your use in changing the default password of the instance to the contents of a user-provided file during its initial startup in the container, as described in Authentication and Passwords. This script, which can be found in dev/Container/ under the InterSystems IRIS installation directory on Linux platforms, is called by the **iris-main --password-file** option, but you can call it in other ways. The script creates a sentinel file that prevents it from running again, so that when the script is invoked during container creation (as by **iris-main**) it will not run every time the container is started.

The changePassword.sh script proceeds as follows:

- If a sentinel file exists in the directory containing the specified password file, the script exits without attempting to change the password.
- If a sentinel file does not exist, the script
    1. Reads the new password from the specified file.
    2. Shuts down the instance if it is running.
    3. Calls **SYS.Container.ChangePassword()** to change the password of all enabled user accounts with at least one role, effectively changing the default password of the instance.
    4. On successful completion of the password change, calls **SYS.Container.ChangeGatewayMgrPassword()** to change the Web Gateway management password to the new password (see Web Gateway Management Pages in the "Web Gateway Operation and Configuration" chapter of the *Web Gate Configuration Guide*).
    5. If the password file is writeable, the script
        - Deletes the password file.
        - Creates a sentinel file.

        If the password file is read-only, no sentinel file is created; this provides compatibility with Docker Secrets, Kubernetes Secrets, and similar technologies.

5.5 Deploying Customized InterSystems IRIS Instances

Every InterSystems IRIS instance, including the one running within an InterSystems IRIS container, is installed with a file in the installation directory named iris.cpf, which contains most of its configuration settings. The instance reads this *configuration parameter file*, or CPF, at startup to obtain the values for these settings. When a setting is modified, the CPF is automatically updated.

However, you may want to deploy multiple instances from the same image but with different configuration settings. You can do this using the **ISC_CPF_MERGE_FILE** environment variable, which lets you specify a separate file containing one or more settings to be merged into the CPF with which a new instance is installed or deployed before the instance is first started. This allows you to deploy multiple instances with differing CPFs from the same source.

For example, the **[config]** section of the CPF included in InterSystems IRIS images contains the default generic memory heap configuration (see Configuring Generic Memory Heap in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*). If you want to increase the size of the generic memory heap for a newly deployed instance, you can use a merge file to change the **[config]/gmheap** setting in the instance's CPF after its container is deployed.

For an example of specifying a merge file when running an InterSystems IRIS container, see the following section. For information about the CPF and the use of the merge feature generally, see Introduction to the Configuration Parameter File in the *Configuration Parameter File Reference*.

Note: The CPF merge feature is part of durable %SYS, and therefore the **ISC_DATA_DIRECTORY** environment variable must also be included in the **docker run** command, as illustrated in the following section.

5.6 Running InterSystems IRIS Containers

This section provides some examples of launching InterSystems IRIS containers with the Docker and **iris-main** options covered in this document, including:

- Command line examples
- Script example
- Docker Compose example

Note: The sample **docker run** commands in this section include only the options relevant to each example and omit options that in practice would be included, as shown (for example) in the sample command in Running an InterSystems IRIS Container with Durable %SYS.

Use of huge pages (see Configuring Large and Huge Pages in the "Vertical Scaling" chapter of the *Scalability Guide*) requires the IPC_LOCK kernel capability. Without this capability, huge pages cannot be allocated when configured for InterSystems IRIS. Most container runtime engines do not grant containers this capability unless it is specifically requested when the container is created. To add the IPC_LOCK capability to a container, include the option **--cap-add IPC_LOCK** in the **docker create** or **docker run** command. This is illustrated in the script example that follows.

5.6.1 Running an InterSystems IRIS Container: Docker Run Examples

The following are examples of **docker run** commands for launching InterSystems IRIS containers using **iris-main** options.

- As described in License Keys for InterSystems IRIS Containers, the required InterSystems IRIS license key must be brought into the container so that the instance can operate. In the example that follows using the **iris-main -key** option as well as the needed options for durable %SYS (see Ensuring that Durable %SYS is Specified and Mounted), the license key is staged in the key/ directory on the volume mounted for the durable %SYS directory — that is, /data/durable/key/ on the external storage, /dur/key/ inside the container — and is copied to the mgr/ directory within the durable %SYS directory (/data/durable/iconfig/mgr/ on the external storage, /dur/iconfig/mgr/ in the container) before the InterSystems IRIS instance is started. Because it is in the mgr/ directory, it is automatically activated when the instance starts.

```
docker run --name iris11 --init --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iris_conf.d
  intersystems/iris:2019.4.0.633.0 --key /dur/key/iris.key
```

- In the next example, which replaces the **--key** option with the **--license-config** option, the license to be served to the instance in the container is identified by its LicenseID field, the instance in the container is specified as a license server, and all of the license keys that can be served by the license server are staged in the licenses/ directory on the volume mounted for the durable %SYS directory — that is, /data/durable/licenses/ on the external storage, /dur/licenses/ inside the container. (The license staging directory cannot be on the local filesystem inside the container.)

  This example also uses specifies a CPF merge file staged on the durable data volume, containing settings to be merged into the InterSystem IRIS instance's CPF (see Deploying Customized InterSystems IRIS Instances) before it is first started. You might use this, for example, to reconfigure the instance's primary and alternate journal directories (**[Journal]/CurrentDirectory** and **AlternateDirectory** in the CPF), which by default are the same directory within the

durable %SYS tree, to be on separate file systems, as described in Separating File Systems for Containerized InterSystems IRIS.

```
docker run --name iris17 --init --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iris_conf.d

  --env ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf intersystems/iris:2019.4.0.633.0
  --license-config "2380451964 iris17,4002,/dur/licenses"
```

- If the instance in the container is a failover member of a mirror, you would add arguments to identify its failover partner, which should also be configured as a license server, using the same licenses:

```
docker run --name iris17 --init --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iris_conf.d

  --env ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf intersystems/iris:2019.4.0.633.0
  --license-config "2380451964 iris17,4002,/dur/licenses iris18,4002,/dur/licenses"
```

  The staging directories, in this case both located on the volume mounted for durable %SYS, should be the same, or contain the same licenses.

- Once container **`iris17`** and the instance inside it are running, you can start another InterSystems IRIS container with a different license to be served to the instance inside it, using the license server configured on **`iris17:4002`** by the first command, as well as a different CPF merge file, as follows:

```
docker run --name iris99 --init --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iris2_conf.d

  --env ISC_CPF_MERGE_FILE=/dur/merge2/merge.cpf intersystems/iris:2019.4.0.633.0
  --license-config "8127394077 iris17,4002"
```

Note: Because the InterSystems IRIS Community Edition image described in Downloading the InterSystems IRIS Docker Image includes a free temporary license, the **--key** and **--license-config** options should not be used with this image.

5.6.2 Running an InterSystems IRIS Container: Script Example

The following script was written to quickly create and start an InterSystems IRIS container for testing purposes. The script incorporates the **iris-main --key** option to copy in the license key, as described in License Keys for InterSystems IRIS Containers.

```
#!/bin/bash
# script for quick demo and quick IRIS image testing

# Definitions to toggle_____________________________________________
container_image="intersystems/iris:2019.4.0.633.0"

# the docker run command
docker run -d
  -p 9091:51773
  -p 9092:52773
  -p 9093:53773
  -v /data/durable:/dur
  -h iris
  --name iris
  --init
  --cap-add IPC_LOCK
  --env ISC_DATA_DIRECTORY=/dur/iris_conf.d
  --env ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf
  $container_image
  --key /dur/key/iris.key
```

5.6.3 Running an InterSystems IRIS Container: Docker Compose Example

Docker Compose, a tool for defining and running multicontainer Docker applications, offers an alternative to command-line interaction with Docker. To use Compose, you create a docker-compose.yml containing specifications for the containers you want to create, start, and manage, then use the **docker-compose** command. For more information, start with Overview of Docker Compose in the Docker documentation.

The following is an example of a compose.yml file. Like the preceding script, it incorporates only elements discussed in this document.

```
version: '3.2'

services:
  iris:
    image: intersystems/iris:2019.4.0.633.0
    command: --license-config "4691540832 iris,4002,/ISC/licenses"
    hostname: iris

    ports:
    # 51773 is the superserver default port
    - "9091:51773"
    # 52773 is the webserver/management portal port
    - "9092:52773"

    volumes:
    - /data/durable:/dur

    environment:
    - ISC_DATA_DIRECTORY=/dur/iris_conf.d
    - ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf
```

5.7 Upgrading InterSystems IRIS Containers

When a containerized application is upgraded or otherwise modified, the existing container is removed or renamed, and a new container is created and started by instantiating a different image with the **docker run** command. Although the purpose is to modify the application, as one might with a traditional application by running an upgrade script or adding a plug-in, the new application instance actually has no inherent association with the previous one. Rather, it is the interactions established with the environment outside the container — for example, the container ports you publish to the host with the **--publish** option of the **docker run** command, the network you connect the container to with the **--network** option, and the external storage locations you mount inside the container with the **--volume** option in order to persist application data — that maintain continuity between separate containers, created from separate images, that represent versions of the same application.

Note: InterSystems IRIS containers of versions prior to 2019.3 that use durable %SYS require a manual procedure before upgrade to 2019.3. For more information, see the 2019.3 *Release Notes*.

5.7.1 Upgrading InterSystems IRIS Containers with Durable %SYS

For InterSystems IRIS, the durable %SYS feature for persisting instance-specific data is used to enable upgrades. As long as the instance in the upgraded container uses the original instance's durable %SYS storage location and has the same network location, it effectively replaces the original instance, upgrading InterSystems IRIS. If the version of the instance-specific data does not match the version of the new instance, durable %SYS upgrades it to the instance's version as needed. (For more information about Durable %SYS, see Durable %SYS for Persistent Instance Data.)

Before starting the new container, you must either remove or stop and rename the original container.

CAUTION: Removing the original container is the best practice, because if the original container is started following the upgrade, two instances of InterSystems IRIS will be attempting to use the same durable %SYS data, which will result in unpredictable behavior, including possible data loss.

Typically, the upgrade command is identical to the command used to run the original container, except for the image tag. In the following **docker run** command, only the *version_number* portion would change between the **docker run** command that created the original container and the one that creates the upgraded container:

```
$ docker stop iris
$ docker rm iris
$ docker run --name iris --publish 9091:51773, 9092:52773, 9093:53773
  --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iconfig
  intersystems/iris:<version_number> --key /dur/key/iris.key
```

5.7.2 Upgrading When Manual Startup is Required

When durable %SYS detects that an instance being upgraded did not shut down cleanly, it prevents the upgrade from continuing. This is because WIJ and journal recovery must be done manually when starting such an instance to ensure data integrity. To correct this, you must use the procedures outlined in Starting InterSystems IRIS Without Automatic WIJ and Journal Recovery in the "Backup and Restore" chapter of the *Data Integrity Guide* to start the instance and then shut it down cleanly. If the container is running, you can do this by executing the command **docker exec -it** *container_name* **bash** to open a shell inside the container and following the outlined procedures. If the container is stopped, however, you cannot start it without automatically restarting the instance, which could damage data integrity, and you cannot open a shell. In this situation, use the following procedure to achieve a clean shutdown before restarting the container:

1. Create a duplicate container using the same command you used to create the original, including specifying the same durable %SYS location and the same image, but adding the **iris-main -up false** option (see The iris-main Program). This option prevents automatic startup of the instance when the container starts.
2. Execute the command **docker exec -it** *container_name* **bash** to open a shell inside the container.
3. Follow the procedures outlined in Starting InterSystems IRIS Without Automatic WIJ and Journal Recovery.
4. When recovery and startup are complete, shut down the instance using **iris stop** *instance_name*. (The instance in a container provided by InterSystems is always named **`IRIS`**.)
5. Start your original container. Because it uses the durable %SYS data that you safely recovered in the duplicate container, normal startup is safe.

6 Additional Docker/InterSystems IRIS Considerations

This section describes some additional considerations to bear in mind when creating and running InterSystems IRIS images container images, including the following:

- Docker Storage Driver
- Locating Image Storage on a Separate Partition

6.1 Docker Storage Driver

Docker supports a number of different storage drivers to manage images. As of this writing, InterSystems supports the **devicemapper** and **overlay2** storage drivers for running InterSystems IRIS in production on Linux hosts, and you must configure the Docker daemon to use one of these drivers. If using **devicemapper**, be sure to read Docker's explanation of using the storage driver in direct-lvm mode for management of container images. To determine which storage driver your OS uses by default and how to change the driver if need be, see Docker storage driver in the Docker documentation and the Compatibility Matrix.

6.2 Locating Image Storage on a Separate Partition

The default storage location for Docker container images is /var/lib/docker. Because this is part of the root file system, you might find it useful to mount it on a separate partition, both to avoid running out of storage quickly and to protect against file system corruption. Both Docker and the OS might have trouble recovering when the above problems emerge. For example, SUSE states: "It is recommended to have /var/lib/docker mounted on a separate partition or volume to not affect the Docker host operating system in case of file system corruption."

A good approach is to set the Docker Engine storage setting to this alternative volume partition. For example, on Fedora-based distributions, edit the Docker daemon configuration file (see Configure and troubleshoot the Docker daemon in the Docker documentation), locate the **ExecStart=** command line option for the Docker Engine, and add **-** as an argument.