Running InterSystems Products in Containers

Version 2020.3
2021-02-04

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

Running InterSystems Products in Containers
InterSystems IRIS Data Platform Version 2020.3 2021-02-04

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 containers and provides the information you need to deploy InterSystems IRIS® and InterSystems IRIS-based applications in containers, using the images provided by InterSystems.

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

You can get a container image for InterSystems IRIS Community Edition, which comes with a free temporary license, from the InterSystems Container Registry or the Docker Store; for more information, see Deploying InterSystems IRIS Community Edition on Your Own System in *Deploy and Explore InterSystems IRIS*.

Important: Looking for basic information about running a container from an InterSystems IRIS image? For an example and brief instructions, please see Run a Container from the InterSystems IRIS Image and the sections that precede it in *First Look: InterSystems Products in Containers*.

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 Containers

Because a 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.

For detailed information about deploying and using InterSystems IRIS in containers, see Running InterSystems IRIS Containers. Some notable features of containerized InterSystems IRIS are briefly described in the following..

- *InterSystems-provided images* — A *container image* is the executable package, while a container is a runtime *instance* of an image. InterSystems provides images containing a fully-installed instance of InterSystems IRIS, as well as other associated images, as described in Using InterSystems IRIS Images.
- *The iris-main program* — The *iris-main* program enables InterSystems IRIS and other products to satisfy the requirements of applications running in 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 also provides a container image that includes both the InterSystems Web Gateway and an Apache web server, providing a web server component for containerized deployments of InterSystems IRIS-based applications; for more information, see Using the InterSystems Web Gateway Container.

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 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 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 image registry of the local host, from which it can be copied to other 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 registry.
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 these 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, and **iris-main** issues a warning when you attempt to do so.

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 Running InterSystems IRIS Containers

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

- Using InterSystems IRIS Images
- The iris-main Program
- Durable %SYS for Persistent Instance Data
- Deploying Customized InterSystems IRIS Instances
- Running InterSystems IRIS Containers
- Upgrading InterSystems IRIS Containers
- Using the InterSystems Web Gateway Container

4.1 Using InterSystems IRIS Images

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

- Container Deployment Platforms Supported by InterSystems

- Installing Docker
- Downloading the InterSystems IRIS Image
- License Keys for InterSystems IRIS Containers
- Security for InterSystems IRIS Containers
- Environment Variables in InterSystems IRIS Containers
- Mirroring with InterSystems IRIS Containers
- Discovering Defaults in InterSystems Images

4.1.1 Container Deployment Platforms Supported by InterSystems

Container images from InterSystems comply with the Open Container Initiative (OCI) specification, and are built using the Docker Enterprise Edition engine, which fully supports the OCI standard and allows for the images to be certified and featured in the Docker Hub registry.

InterSystems images are built and tested using the widely popular container Ubuntu operating system, and are therefore supported on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.

Note: 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 Desktop (for Windows and MacOS) creates Linux VMs, running under the respective platform virtualizers, to host containers. These additional layers add complexity that prevents InterSystems from supporting Docker Desktop at this time.

We understand, however, that for testing and other specific purposes, you may want to run InterSystems IRIS-based containers from InterSystems on Windows or MacOS. 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.

4.1.2 Installing Docker

The Docker Engine consists of an open source containerization technology combined with a workflow for building and running containerized applications. 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.

4.1.3 Downloading the InterSystems IRIS Image

The InterSystems Container Registry (ICR) at https://containers.intersystems.com/ includes repositories for all images available from InterSystems, including InterSystems IRIS images. Using the InterSystems Container Registry describes the images available from the ICR and explains how to use your WRC credentials to authenticate to the registry so you can download them.

Note: You can also download an InterSystems IRIS Community Edition image, which comes with a free built-in 13-month license (as well as some limitations), from the ICR without authenticating. The Community Edition image is also available on the Docker Store's InterSystems IRIS Data Platform page. For more information, see Deploying InterSystems IRIS Community Edition on Your Own System in *Deploy and Explore InterSystems IRIS*.

Your organization may already have an InterSystems IRIS image available for your use in its own private image registry; if so, obtain the location and the credentials needed to authenticate from the responsible administrator. Once you are logged into either the ICR or your organization's registry, you can use the **docker pull** command to download the image; the following example shows a pull from the ICR:

```
$ docker login containers.intersystems.com
Username: pmartinez
Password: **********
$ docker pull containers.intersystems.com/intersystems/iris:2020.3.0.221.0
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
$ docker images
REPOSITORY                                      TAG             IMAGE ID      CREATED      SIZE
containers.intersystems.com/intersystems/iris 2020.3.0.221.0 15627fb5cb76  1 minute ago 1.39GB
containers.intersystems.com/intersystems/sam  1.0.0.115       15627fb5cb76  3 days ago   1.33GB
acme/centos                                     7.3.1611        262f7381844c  2 weeks ago  192MB
acme/hello-world                                latest          05a3bd381fc2  2 months ag  1.84kB
```

For simplicity, the instructions in this document assume you are working with the InterSystems IRIS image **`intersystems/iris:2020.3.0.221.0`**.

4.1.4 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 described in the previous section comes with a special free temporary license. Generally, however, license keys are not 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 and 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.

4.1.5 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.) Operating system-based authentication is enabled on the instance, which means that **`irisowner`** can connect to the instance without authentication. Because commands issued from outside the container using **docker exec** are executed inside the container as **`irisowner`**, you can use this method to connect to an instance inside a container without authentication, for example executing **docker exec -it** *container_name* **iris terminal IRIS** to open the InterSystems Terminal without being prompted for credentials.

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 information about the installation parameters used to specify these configuration details, see Environment Variables in InterSystems IRIS Containers. For more information on these general installation-related topics, see InterSystems IRIS Installation in the "Installing on UNIX®, Linux, and macOS" chapter of the *Installation Guide*.

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**. For effective security, it is important that this default password be changed immediately upon deployment of your container, which can be done using one of the following approaches. Either of these methods can be incorporated into automated deployment.

Important: Once the instance is deployed and running with the new default password, you should log into each of the predefined accounts, which are configured to require a password change on first login, so that they are all secured with new individual passwords of your choosing rather than sharing a password; as an alternative, you can also disable one or more of them.

CAUTION: If you *do not* use one of the methods listed here to modify the default password, it is critical that you either log into each predefined account and change the password or disable the accounts as soon as possible.

- The PasswordHash CPF setting

  During automated deployment of InterSystems IRIS on UNIX and Linux platforms, you can change the default password for one or more instances before they are started using the configuration parameter file (CPF) PasswordHash setting in conjunction with the CPF merge feature.

  Rather than recording the plain-text password for each account (a security risk), InterSystems IRIS stores only an irreversible cryptographic hash of that password; when the user logs in, the password value entered is hashed using the same algorithms, and the two versions are compared to authenticate the user. For more information about the stored password hash, see InterSystems Authentication in the "Authentication" chapter of the *Security Administration Guide*.

  You can set or change the stored password hash (and thus the password) for all of an instance's enabled user accounts that have at least one assigned role using the PasswordHash setting, which is in the **[Startup]** section of the CPF. When included in a CPF merge file at deployment (on UNIX® and Linux systems only, see Deploying Customized InterSystems IRIS Instances), this setting customizes the default password of the predefined accounts on the instance (except

CSPSystem, which does not have an assigned role), replacing **SYS** with the password of which the provided value is a hash.

Important: Once you have changed the passwords of the predefined accounts from the new default password (which you should do immediately after deployment, as advised above), edit the iris.cpf file and remove the value for the PasswordHash setting. If you do not do this, the next time you start or restart the instance, the password for all of the instance's enabled user accounts that have at least one assigned role will be changed to the password represented by the PasswordHash setting.

PasswordHash specifies a hashed password and the salt for the password. A description of the algorithms used to convert a plain-text password to these values is contained in InterSystems Authentication in the "Authentication" chapter of the *Security Administration Guide*, and tools for applying them are available in the %SYSTEM.Encryption API. However, undertaking this conversion as a manual procedure is not recommended, as it is likely to be error-prone and time-consuming. For your convenience, InterSystems publishes the image for a nanocontainer, **passwordhash**, that does this conversion for you. The following is an example of using this container:

```
docker run --rm -it intersystems/passwordhash:1.1
Enter password: <password_entry>
Enter password again: <password_entry>
PasswordHash=dd0874dc346d23679ed1b49dd9f48baae82b9062,vatv5ja7
```

You would then copy and paste the output and place it in your CPF merge file as follows:

```
[Startup]
PasswordHash=dd0874dc346d23679ed1b49dd9f48baae82b9062,vatv5ja7
```

After deployment, the default password for the predefined accounts (other than **CSPSystem**) is *password_entry*.

Important: The use of the PasswordHash setting to change passwords after deployment is not recommended; the Editing an Existing User section in the "Users" chapter of *Security Administration Guide* describes how to change an existing user's password.

Note: Blank passwords cannot be used with the PasswordHash setting.

- The password change script

  The password change 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. 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. Makes an API call 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, makes another API call 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*).

Note: Only the Web Gateway management password for the local instance is affected by this call; any Web Gateway containers in use (see Using the InterSystems Web Gateway Container) are unaffected.

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.

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.

4.1.6 Environment Variables in InterSystems IRIS Containers

There are a number of installation parameters available for use in configuring unattended installation of InterSystems IRIS instances on UNIX and Linux (see Unattended InterSystems IRIS Installation in the *Installation Guide*). The installation parameters in the following table are used in the creation of InterSystems IRIS images by InterSystems and thus are set as variables, to the values shown, in all containers created from such images.

Table 1: Installation Parameters Set as Environment Variables in InterSystems IRIS Containers

Parameter/Variable	Description	InterSystems Value
ISC_PACKAGE_INSTANCE-NAME	Name of the instance.	IRIS
ISC_PACKAGE_INSTALLDIR	Directory in which the instance is installed.	/usr/irissys
ISC_PACKAGE_IRISUSER	Effective user for the InterSystems IRIS super-server.	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

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

4.1.7 Mirroring with InterSystems IRIS Containers

InterSystems IRIS instances deployed in containers can be configured as mirrors the same way you would configure those deployed from a kit, using the procedures described in Configuring Mirroring in the *High Availability Guide*. However, there are a few points to bear in mind:

- The arbiter configured for the mirror (as strongly recommended by InterSystems) can be deployed from the arbiter image provided by InterSystems, which you can download using the same procedures described for the InterSystems IRIS image. You can also use a noncontainerized InterSystem IRIS instance or an arbiter installed from a kit.

- When you deploy the InterSystems IRIS and arbiter containers, you must ensure that you publish the ports used by the mirror members and arbiter to communicate with each other, as described in Mirror Member Network Addresses in the *High Availability Guide*.
- Be sure to review Mirroring Communication, Sample Mirroring Architecture and Network Configurations, and Locating the Arbiter to Optimize Mirror Availability in the *High Availability Guide* to ensure that your deployment addresses all of the needed networking considerations.

4.1.8 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:2020.3.0.221.0
  "Labels": {
      "com.intersystems.adhoc-info": "", "com.intersystems.platform-version": "2020.3.0.221.0",
      "com.intersystems.ports.default.arbiter": "2188",
      "com.intersystems.ports.default.license-server": "4002",
      "com.intersystems.ports.default.superserver": "1972",
      "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 Aug 16 2020 00:37:59 EST",
      "com.intersystems.product-timestamp.iso8601": "2020-08-16T05:37:59Z",
      "maintainer": "InterSystems Worldwide Response Center <support@intersystems.com>",
      "org.opencontainers.image.created": "2020-08-16T07:57:10Z",
      "org.opencontainers.image.documentation": "https://docs.intersystems.com/",
      "org.opencontainers.image.title": "intersystems/iris",
      "org.opencontainers.image.vendor": "InterSystems",
      "org.opencontainers.image.version": "2020.3.0.221.0"
      }
```

4.2 The iris-main Program

There are several requirements an application must satisfy in order to run in a 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.

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 10 seconds, and then send the SIGKILL signal. Kubernetes operates in a similar fashion. The **iris-main** program intercepts SIGTERM and SIGINT and executes a graceful shutdown of IRIS.

Important: If your IRIS instance is particularly busy when the **docker stop** command is issued, 10 seconds may not be long enough to bring it to a complete stop, which may result in Docker sending a SIGKILL. SIGKILL cannot be trapped or handled, and is similar to powering off a machine in terms of program interruption and potential data loss. If your IRIS container receives a SIGKILL, on the next start it will engage in normal IRIS crash recovery procedures. To prevent this, use the **--time** option with your **docker stop** command, or the terminationGracePeriodSeconds value in your Kubernetes configuration, to specify a wait time longer than 10 seconds.

- 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
  ```

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

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.

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
-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

Option	Description	Default
-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	none
-t *command* --terminate=*command*	Execute a custom shell command after any other arguments are processed	none
-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, which makes use of the password change script described in Authentication and Passwords, 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.	none
--version	Prints the **iris-main** version	N/A
-h, --help	Displays usage information and exits	N/A

4.3 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.

4.3.1 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 — con-

tainerization 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.

4.3.2 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*)

  Note: Web Gateway containers also have a durable %SYS feature, and when one is in use (see Using the InterSystems Web Gateway Container), this directory is located on its own persistent storage.

- The file /httpd/httpd.conf, the configuration file for the instance's private web server. ("Supported Web Servers" in the "Supported Technologies" chapter of 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.

4.3.3 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.

InterSystems recommends specifying a subdirectory of the mounted volume as the durable %SYS location. For example, if an external file system location is mounted as the volume /external in the container, /external should not be specified as the durable %SYS location, but rather a directory on /external such as /external/durable.

4.3.4 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:2020.3.0.221.0
```

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, and **iris-main** issues a warning when you attempt to do so. A similar warning is issued if the specified durable %SYS location is on a mounted volume that has a file system type of **`cifs`** or any type containing the string **`fuse`**.

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.

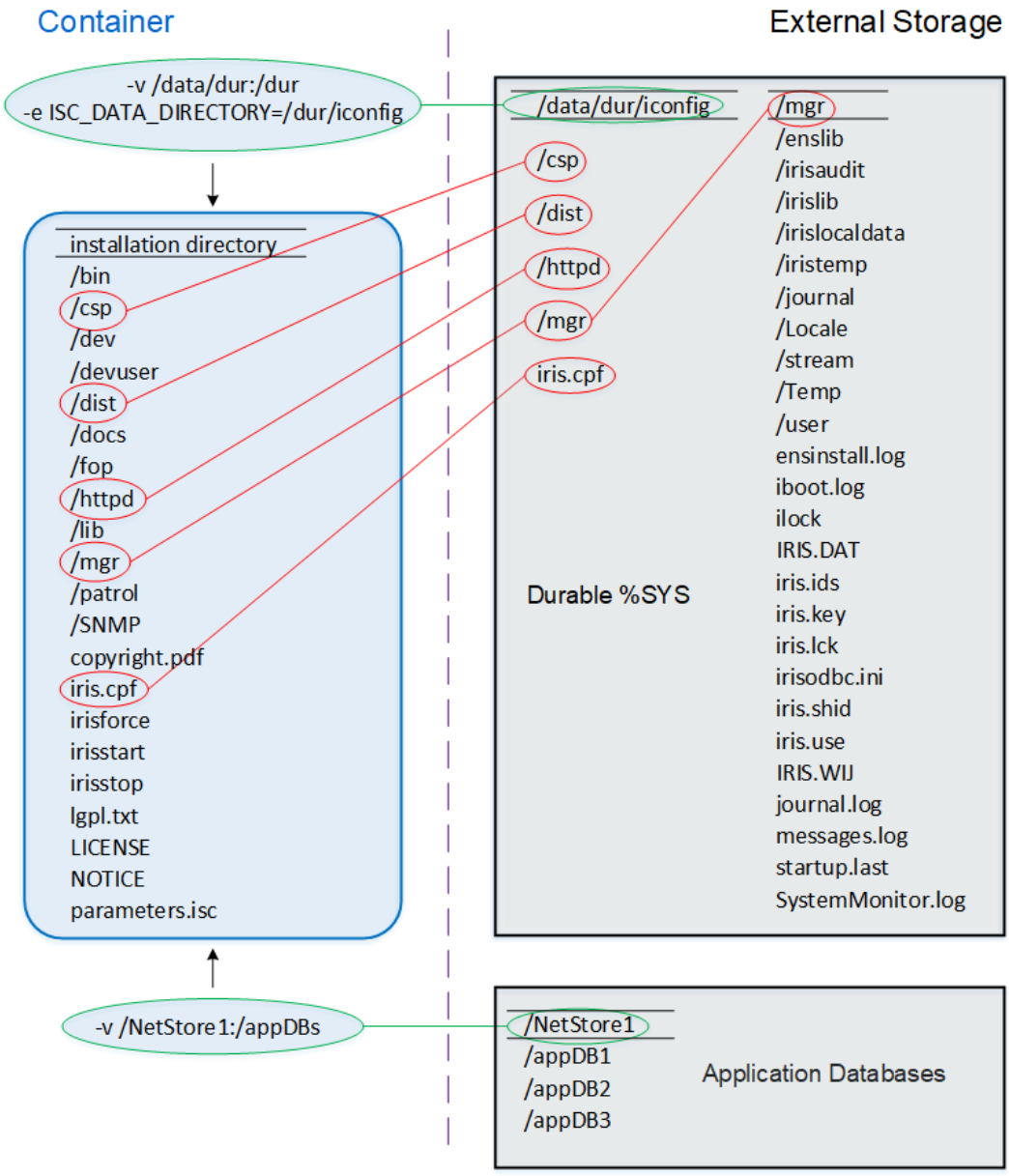
  If for any reason the process of copying the durable %SYS data and resetting internal pointers fails, the durable %SYS directory is marked as incomplete; if you try again with the same directory, the data in it is deleted before the durable %SYS process starts.
- If the durable %SYS directory specified 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



4.3.5 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)**.

4.3.6 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.

4.3.7 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 "Preparing to Install InterSystems IRIS" chapter of the *Installation Guide* for more information about separation of file systems for InterSystems IRIS.

4.4 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 *config-*

uration 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. On UNIX® and Linux systems ou can do this using the CPF merge feature, in which you use **ISC_CPF_MERGE_FILE** environment variable to 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 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 CPF 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*.

4.5 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.

4.5.1 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:2020.3.0.221.0 --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:2020.3.0.221.0
  --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:2020.3.0.221.0
  --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:2020.3.0.221.0
  --license-config "8127394077 iris17,4002"
```

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

4.5.2 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-arm64:2019.4.0.633.0"

# the docker run command
docker run -d
  -p 9091:1972
  -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
```

4.5.3 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:2020.3.0.221.0
    command: --license-config "4691540832 iris,4002,/ISC/licenses"
    hostname: iris

    ports:
    # 1972 is the superserver default port
    - "9091:1972"
    # 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
```

4.6 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*.

4.6.1 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 9091:1972, 9092:52773, 9093:53773
  --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iconfig
  intersystems/iris:<version_number> --key /dur/key/iris.key
```

4.6.2 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.

4.7 Using the InterSystems Web Gateway Container

The InterSystems IRIS Web Gateway provides the communications layer between the hosting web server and InterSystems IRIS data platform for any InterSystems IRIS web application, and is typically installed on systems acting as web servers within InterSystems IRIS configurations. Procedures for manually configuring the Web Gateway for one or more InterSystems IRIS instances are provided in Web Gateway Operation and Configuration and Using Web Applications with a Remote Web Server in the *Web Gateway Configuration Guide*, which fully covers the use and operation of the Web Gateway with compatible web servers. The Web Gateway configuration for an installed InterSystems IRIS instance is contained in the file *install-dir*/CSP/bin/CSP.ini.

The **webgateway** image available from InterSystems, which includes both the Web Gateway and an Apache web server, provides a web server component for containerized deployments of InterSystems IRIS-based applications. Automated deployment of the Web Gateway container can be effected by using manual procedures to develop a CSP.ini file containing the desired Web Gateway configuration, as well as apache2.load and apache2.conf files specifying the desired Apache web server modules and configuration, and placing them in their expected locations in the Web Gateway container (/opt/webgateway/bin/ and /etc/apache2/mods-available/, respectively) as part of the deployment process. The durable %SYS feature and the Web Gateway's entrypoint application /cspEntryPoint.sh provide a simple way to do this, as the script automatically links these files (plus the CSP.log log file) to the corresponding files on the specified durable %SYS volume. For example, suppose you place your CSP.ini, apache2.load, and apache2.conf files in the /config.d directory on the container host and use the following **docker run** command:

```
docker run --name webserver01 --init --volume /host/config.d:/config.d
  --env ISC_DATA_DIRECTORY=/config.d intersystems/webgateway:2020.3.0.633.0
```

As a result, through durable %SYS, the file /opt/webgateway/bin/CSP.ini in the container file system would be linked to the file /host/config.d/CSP.ini on the container host's file system, and so on for the other files. If one of the files does not exist on the mounted volume, for example CSP.log, it is copied from its container location to the durable %SYS directory and the link is then created.

Important: Under the arrangement described here, to maintain the ability to update the Web Gateway configurations in a containerized deployment by pushing a new version of the CSP.ini file to the container hosts and thereby to the containers, you must relocate Web Gateway-maintained version/timestamp parameters from CSP.ini to the Web Gateway runtime parameters file, CSPRT.ini, and set the READONLY parameter in the mounted CSP.ini file so that the Web Gateway cannot modify it at runtime. For information about how to do this, see Read-only Option for Automated Deployment Sites in the *Web Gateway Configuration Guide*.

5 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:

- Locating Image Storage on a Separate Partition
- Docker Bridge Network IP Address Range Conflict

5.1 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.

5.2 Docker Bridge Network IP Address Range Conflict

For container networking, Docker uses a bridge network (see Use bridge networks in the Docker documentation) on subnet 172.17.0.0/16 by default. If this subnet is already in use on your network, collisions may occur that prevent Docker from starting up or generate network errors.

To resolve this, you can edit the bridge network's IP configuration in the Docker configuration file to reassign the subnet to a range that is not in conflict with your own IP addresses (your IT department can help you determine this value). To make this change, add a line like the following to the Docker daemon configuration file, which is /etc/docker/daemon.json by default:

```
"bip": "192.168.0.1/24"
```

Detailed information about the contents of the daemon.json file can be found in Daemon configuration file in the Docker documentation; see also Configure and troubleshoot the Docker daemon.