Running InterSystems Products in Containers

Version 2024.2
2024-09-05

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

Running InterSystems Products in Containers
PDF generated on 2024-09-05
InterSystems IRIS® Version 2024.2

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

List of Figures

List of Tables

Running InterSystems Products in Containers

This document 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.

To learn how to *quickly get started running an InterSystems IRIS container*, including downloading the needed image, see InterSystems IRIS Basics: Run an InterSystems IRIS Container; for a quick, online hands-on exercise, see Deploying and Customizing InterSystems IRIS 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 Docker Hub; for more information, see Deploying InterSystems IRIS Community Edition on Your Own System.

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.

2 The Docker Container Platform

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.

3 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 Using 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 a range of images containing a fully-installed instance of InterSystems IRIS, as described in Using InterSystems IRIS Images; this includes images for InterSystems IRIS with IntegratedML and InterSystems IRIS for Health, as well as free Community Edition instances. 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 Images.
- *Secure containers* — All InterSystems IRIS images contain a nonroot instance, which means that it was installed by user **`irisowner`**, which does not have root privileges and holds all ownership; nothing is therefore owned by root, or owned or run as any user but **`irisowner`**. When deploying InterSystems IRIS under the strictest security, you can use a *locked down* image to deploy a containerized instance that in addition to being nonroot was installed with Locked Down security. For more information, see Security for InterSystems IRIS Containers.
- *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.

- *Container image catalog* — In addition to InterSystems IRIS images, InterSystems provides the following associated images:
  - **`webgateway`** — Deploys both the InterSystems Web Gateway and a web server, providing a web server component for containerized deployments of InterSystems IRIS-based applications; for more information, see Web Access Using the Web Gateway Container.
  - **`arbiter`** — Deploys an arbiter node as part of mirrored deployments; for more information, see Mirroring with InterSystems IRIS Containers.
  - **`iris-operator`** — Deploys the InterSystems Kubernetes Operator (IKO), which extends the open-source Kubernetes container orchestration engine with a custom resource called *IrisCluster*, representing an InterSystems IRIS sharded cluster, distributed cache cluster, or standalone instance.
  - **`iam`** — Deploys InterSystems API Manager (IAM).
  - **`sam`** — Deploys InterSystems System Alerting and Monitoring (SAM).
  - **`iscreports_server`** — Deploys InterSystems Reports Server.
  - **`passwordhash`** — Converts a plain-text password to the hashed version, with salt, required to use the InterSystems IRIS PasswordHash configuration parameter; for more information, see Authentication and Passwords.

  For a description of the InterSystems Container Registry and information about listing and downloading the images it contains, see Using the InterSystems Container Registry.

4 Container Basics

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

- Container Contents
- The Container Image
- Running a Container

4.1 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.

4.2 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 the Ubuntu operating system 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 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.

4.3 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.)

CAUTION: A container's host machine must satisfy the minimum supported CPU requirement for InterSystems products. If the host machine does not meet this requirement, the InterSystems IRIS container does not start.

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**.

5 Using InterSystems IRIS 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:

- Automated Deployment of InterSystems IRIS Containers
- Using InterSystems IRIS Images
- The iris-main Program
- Durable %SYS for Persistent Instance Data
- Web Access Using the Web Gateway Container
- Running InterSystems IRIS Containers
- Upgrading InterSystems IRIS Containers
- Creating InterSystems IRIS Images
- InterSystems IRIS Containerization Tools

5.1 Automated Deployment of InterSystems IRIS Containers

Containers lend themselves to automated deployment in many ways. InterSystems IRIS data platform provides two methods for automated deployment of multicontainer topologies (such as sharded clusters, distributed cache clusters, and mirrors) that are fully operational following deployment. Advanced methods include the following:

- The InterSystems Kubernetes Operator

  Kubernetes is an open-source orchestration engine for automating deployment, scaling, and management of containerized workloads and services. You define the containerized services you want to deploy and the policies you want them to be governed by; Kubernetes transparently provides the needed resources in the most efficient way possible, repairs or restores the deployment when it deviates from spec, and scales automatically or on demand. The InterSystems Kubernetes Operator (IKO) extends the Kubernetes API with the *IrisCluster* custom resource, which can be deployed as an InterSystems IRIS sharded cluster, distributed cache cluster, or standalone instance (all optionally mirrored) on any Kubernetes platform. The IKO also adds InterSystems IRIS-specific cluster management capabilities to Kubernetes, enabling automation of tasks like adding nodes to a cluster, which you would otherwise have to do manually by interacting directly with the instances.

  For more information on using the IKO, see Using the InterSystems Kubernetes Operator.

- Configuration merge

  The configuration merge feature, available on Linux and UNIX® systems, lets you vary the configurations of InterSystems IRIS containers deployed from the same image (or local instances installed from the same kit) by simply applying the desired declarative configuration merge file to each instance in the deployment.

This merge file, which can also be applied to an existing instance, updates an instance's *configuration parameter file* (CPF), which contains most of its configuration settings; these settings are read from the CPF at every startup, including the first one after an instance is deployed. When you apply configuration merge during deployment, you in effect replace the default CPF provided with the instance with your own updated version. This enables you to deploy containers with varying configurations from the same image, or install differently-configured instances from the same kit, directly into a multi-instance topology, rather than having to configure the instances into the desired topology after deployment. For example, in automated containerized deployment of a sharded cluster with compute nodes, you can apply different merge files for data node 1, the remaining data nodes, and the compute nodes in that order; when all of the instances are up and running, so is the sharded cluster. In similar fashion, when deploying a mirror, you would apply different configuration merge files for the primary, backup, and async members. Even a mirrored sharded cluster is easily deployed using this approach.

The IKO, described in the preceding sections, incorporates the configuration merge feature.

For information about using configuration merge, including examples of use cases for automated deployment using configuration merge, see Automating Configuration of InterSystems IRIS with Configuration Merge.

Important: When a container is deployed with configuration merge using the **`ISC_CPF_MERGE_FILE`** environment variable to specify the merge file, that file is continuously monitored for updates, which are immediately merged when they occur, as long as the container is running. This means that you can update the configuration of a containerized instance at any time simply by updating the merge file. For more information, see How do I reconfigure an existing instance using configuration merge? in *Automating Configuration of InterSystems IRIS with Configuration Merge*.

5.2 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
- Discovering Defaults in InterSystems Images
- License Keys for InterSystems IRIS Containers
- Security for InterSystems IRIS Containers
- Mirroring with InterSystems IRIS Containers

5.2.1 Container Deployment Platforms Supported by InterSystems

Container images from InterSystems comply with the Open Container Initiative (OCI) specification and are therefore supported on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.

Note: The specific instructions and procedures in this document are intended to be used with Docker 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.

5.2.2 Installing the Container Runtime Engine

Install the container runtime engine(s) on which you intend to deploy InterSystems containers, such as Docker or Podman, on your servers.

5.2.3 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 Docker Hub. 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 in to 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:latest-em
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 2023.2.0.299.0 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
```

Note: The image tags shown in this document are examples only. Please go to the InterSystems Container Registry (ICR) to browse current repositories and tags.

5.2.4 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:latest-em
  "Labels": {
      "com.intersystems.adhoc-info": "",
      "com.intersystems.platform-version": ":2023.2.0.299.0",
      "com.intersystems.ports.default.arbiter": "2188",
      "com.intersystems.ports.default.license-server": "4002",
      "com.intersystems.ports.default.superserver": "1972",
      "com.intersystems.product-name": "IRIS",
      "com.intersystems.product-platform": "dockerubuntux64",
      "com.intersystems.product-timestamp": "Wed May 16 2022 00:37:59 EST",
      "com.intersystems.product-timestamp.iso8601": "2023-08-16T05:37:59Z",
      "maintainer": "InterSystems Worldwide Response Center <support@intersystems.com>",
      "org.opencontainers.image.created": "2023-08-16T07:57:10Z",
      "org.opencontainers.image.documentation": "https://docs.intersystems.com/",
      "org.opencontainers.image.title": "intersystems/iris",
      "org.opencontainers.image.vendor": "InterSystems",
      "org.opencontainers.image.version": "2023.2.0.299.0"
      }
```

5.2.5 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 Managing InterSystems IRIS Licensing.

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, offers an option for handling the license key that can be used in a **docker start** or **docker run** command starting an InterSystems IRIS container. 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, often the durable %SYS volume. 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 **--key** option allows you to update an instance's license without having to upgrade the container. When the option is used in a **docker run** or **docker start** command with an InterSystems IRIS image, **iris-main** continuously monitors the staged license key for changes (assuming it remains in its original location); if any change in the file is detected, it is copied to the current mgr/ directory and a **%SYSTEM.License.Upgrade()** API call is made.

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

Important: When using core-based IRIS licenses, the number of cores your container runtime engine makes available to your InterSystems IRIS container must be equal to or less than the number of cores in your license. If your host has more cores, you must restrict the CPUs used by the container when you start it using the --cpuset-cpus and --cpus options. For an example involving InterSystems IRIS Community Edition instances, the free license for which is limited to 20 cores, see Deploy InterSystems IRIS Community Edition on Your Own System.

5.2.6 Security for InterSystems IRIS Containers

When working with InterSystems IRIS images from InterSystems it is important to understand the security-related mechanisms and options, including:

- Ownership and directories
- Authentication and passwords
- Locked down InterSystems IRIS container

Important: While InterSystems IRIS security is comprehensive and container images from InterSystems are engineered to support the strictest security requirements, there are some important security practices designed for noncontainerized instances to which containerized instances present a challenge. In particular, operations requiring human intervention are not suitable for automated containerized deployment and operation. For example, startup with interactive key activation (which is the default behavior for an instance with a database encryption key activated) interrupts startup with a prompt for the location of the key, to which an operator must respond. Other tasks, such as configuring an instance to use delegated authorization, are typically done using the interactive ^SECURITY routine.

The configuration merge feature can help in some cases by allowing you to automatically deploy containerized instances with any desired configuration, but some security tasks are best addressed by using the features of container orchestrators such as Kubernetes, which rely on security mechanisms designed for containerized software. For example, *secrets* are a widely used mechanism in which a small amount of sensitive information is placed in an object that can then be provided by the platform when needed, letting you avoid the inclusion of confidential data in application code; this would be a suitable and safe means of providing an encryption key to a containerized InterSystems IRIS instance configured for startup with unattended key activation, which does not require human intervention. In addition, third-party tools, such as Hashicorp's Vault, work with Kubernetes and other platforms to provide additional support for managing and safeguarding sensitive data with policies and multiple options to fit a range of needs.

Note: For important information about security for Web Gateway containers, see Web Gateway Container Security.

Ownership and Directories

The InterSystems IRIS instance in a container created from an InterSystems image is always named **`IRIS`** and is nonroot, meaning it was installed and is owned by a user account, **`irisowner`** (UID 51773), which does not have root privileges. All file system entities comprising the instance are owned by **`irisowner`**, and nothing is therefore owned by root. Operating system-based authentication is enabled on the instance, which means that an **`irisowner`** process or client authenticated on another system 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 command to conveniently connect to the instance without authentication. For example, to open the InterSystems Terminal for an instance in a container named **`iris273`** without being prompted for credentials, you could use the following command:

```
docker exec -it iris273 iris terminal IRIS
```

The installation directory (containing the mgr/ subdirectory) within the container is /usr/irissys/. The working directory (containing files such as iris-main.log) is /home/irisowner/, while the registry directory is /home/irisowner/irissys/.

For information about the installation parameters used to specify these configuration details, see Required Environment Variables. For more information on these general installation-related topics, see Install on UNIX®, Linux, and macOS.

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

Important: When using the durable %SYS feature to provide a containerized InterSystems IRIS instance with persistent storage, the host file system location mounted and specified for this purpose must be writable by user 51773. (You will most likely need root privileges to effect this.)

When using durable %SYS with an InterSystems IRIS container deployed with the Pod Manager tool (podman), you must do the following:

- Issue the following command before starting the container:

  ```
  podman unshare chown 51773:51773 $INSTANCEDIR
  ```

  where $INSTANCEDIR is the host file system location that will contain the durable %SYS directory.
- If Red Hat Security-Enhanced Linux (SELinux) is active, include the **--privileged=true** flag when creating the container.

When using durable %SYS on Kubernetes *without* the InterSystems Kubernetes Operator, you must include the following security context setting in the pod specification:

```
securityContext:
  fsGroup: 51773
```

Before upgrading an InterSystems IRIS container that uses durable %SYS to version 2021.2 or later, you *must* make the existing durable directory writable by user 51773.

Note: If the **`irisowner`** user account is not defined in the /etc/passwd file on the system hosting the container, it is represented by its UID (51773) on that system.

Authentication and Passwords

OS-based authentication (see Operating System–Based Authentication) 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).

As with non-containerized instances, you can configure user accounts within a containerized instance so that they can use escalation roles. See Escalation Roles.

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. Any of these methods can be incorporated into automated deployment.

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

Only the **iris-main --password-file** option, described below, changes the password (**SYS**) for the predefined **`CSPSystem`** account, which is often configured in the InterSystems Web Gateway as the account used to authenticate to the InterSystems IRIS instances in its server access profiles. If you do not use the **--password-file** option in particular to change the default password, you must log in to the **`CSPSystem`** account and change the password as soon as possible after deployment. For more information about configuring Web Gateway authentication, see Web Gateway Container Security.

Important: Once the instance is deployed and running with the new default password, you should log in to 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.

Note: 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.

- 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 first started using the configuration parameter file (CPF) PasswordHash setting in conjunction with the configuration 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 Instance Authentication.

  You can set or change the stored password hash (and thus the password) for all of a newly-deployed instance's predefined accounts (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 configuration merge file at deployment (on UNIX® and Linux systems only), 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.

  Immediately after deployment, as noted above, you should individually change the passwords of the predefined accounts from the new default password set by PasswordHash. The PasswordHash parameter works just once for a given instance, and can therefore be left in an instance's CPF without having any effect.

  The arguments to the PasswordHash parameter are a hashed password and its salt, and optionally the hashing algorithm and work factor used to hash the password (the defaults for the latter are SHA512 and 10000, respectively). All of these arguments are shown in the following example:

```
[Startup]
PasswordHash=0fadb1a665e04ef1bfe929da8457466883e0a3a42c1a34acec49dbc1ff8c4c40f184659ce180c10898db836,db874dc346d2679ed1b49db9f48baae82b962,10000,SHA512
```

  A description of the algorithms used to convert a plain-text password to these values is contained in Instance Authentication, 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, the InterSystems Container Registry (described in Downloading the InterSystems IRIS Image) provides the image for a nanocontainer, **passwordhash**, that does this conversion for you and displays the result in the context of the PasswordHash parameter. You can optionally specify the workfactor and algorithm you want to use; if not, the default are used. The following is an example of using this container:

```
$ docker run --rm -it containers.intersystems.com/intersystems/passwordhash:1.1 -algorithm SHA512
 -workfactor 10000
Enter password:
Enter password again:
PasswordHash=0fadb1a665e04ef1bfe929da8457466883e0a3a42c1a34acec49dbc1ff8c4c40f184659ce180c10898db836,db874dc346d2679ed1b49db9f48baae82b962,10000,SHA512
```

  You would then copy and paste the output and place it in the [Startup] section of your configuration merge file as shown above. After deployment, the default password for the predefined accounts (other than **CSPSystem**) is what you entered at the prompts.

  Note: You can display usage information using the **iris-main --help** option as shown:

```
$ docker run containers.intersystems.com/intersystems/passwordhash:1.1 --help
Usage of /passwordhash:
  -algorithm string
        Pseudorandom function to use (default "SHA512")
  -workfactor int
        PBKDF2 Work Factor (default 10000)
```

You can also provide the password to be hashed as input in the command, for example:

```
$ echo **** | docker run --rm -i containers.intersystems.com/intersystems/passwordhash:1.1
```

Important: The PasswordHash property can be used just once on any given InterSystems IRIS instance, and only if the default password has not yet been changed for any of the predefined accounts. Because allowing the default password to remain unchanged following deployment is a serious security risk, the PasswordHash setting should be used in a configuration merge operation to change the default password during deployment and not later. (For information on how to change an individual user's password, see Edit an Existing User Account.)

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

- The **iris-main --password-file** option

  This option to the **iris-main** entrypoint application changes the default password of an InterSystems IRIS instance's predefined accounts, *including* the **CSPSystem** account, to the contents of a user-provided file during its initial startup in the container, then deletes the password file and creates a sentinel file that prevents it from running again, so that the option will not be invoked every time the container is started. In details, the following steps are taken:

  – 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 password of the predefined **CSPSystem** account (as described earlier in this section).
    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.

       The **iris-main --password-file** option invokes a script, changePassword.sh, which can be found in dev/Container/ under the InterSystems IRIS installation directory on Linux platforms (including within an InterSystems-provided **iris** container). You can call this script in other ways in order to integrate it into your own tools.

  For information about the **--password-file** option, see The iris-main Program.

- The SYS.Container API

  InterSystems IRIS is distributed with an API call, **SYS.Container.ChangePassword()**, that is also useful in scripts and other automation. **SYS.Container.ChangePassword()** changes the password of all of an instance's enabled user accounts that have at least one assigned 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, and is called by the changePassword.sh

script and thus by the **iris-main --password-file** option. When using it, bear in mind the risks of committing the password to a file for any significant length of time.

The API also includes the **SYS.Container.ChangeGatewayMgrPassword()** call (also called by the script) which changes the password of the **`CSPSystem`** account on both the InterSystems IRIS instance and the local Web Gateway, if any.

For information about the SYS.Container API, see SYS.Container API.

Locked Down InterSystems IRIS Container

To support the strictest security requirements, InterSystems provides an image named **`iris-lockeddown`**, from which you can deploy a highly secure InterSystems IRIS container. The differences between containers from this image and those from the standard **`iris`** image are detailed here.

Note: The characteristics of the **`iris-lockeddown`** image are subject to change as best practices evolve. We may add, remove, or change features in response to our best understanding of current security practices and the requirements of the production container orchestrators in use by our customers.

- The instance in a locked down InterSystems IRIS container was installed with Locked Down security, as opposed to the Normal security installation of an instance in the standard InterSystems IRIS container. For details on the differences between Locked Down and Normal security, see Prepare for InterSystems Security in *Securing Your Instance*.
- If InterSystems System Alerting and Monitoring (SAM) is deployed with the instance, you must give it access to the instance by changing the **Allowed Authentication Method** setting of the **/api/monitor** web application from **Password** to **Unauthenticated**. To do this, on the Web Applications page of the Management Portal (**System Administration > Security > Applications > Web Applications**), click **/api/monitor** in the left-hand column to display the Edit Web Application page, make the needed change in the **Security Settings** section, and click **Save**.
- In addition to the environment variables defined in the standard container, as listed in the following section, the **`SYS_CONTAINER_LOCKEDDOWN`** variable is defined as **`1`** in a locked down container.

5.2.7 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. 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. (Note that this is a nonroot image, as described in Security for InterSystems IRIS Containers.) You can also use a noncontainerized InterSystems 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, their ISCAgents, and the arbiter to communicate with each other, as described in Mirror Member Network Addresses.
- The ISCAgent on each failover and DR async mirror member must be configured to start automatically before InterSystems IRIS starts. This is the default behavior for InterSystems IRIS images. When running an InterSystems IRIS container, you can use the following **iris-main** ISCAgent options:

  --ISCAgent true|false

  If `true`, the ISCAgent starts when the container starts on the on the default ISCAgent port, 2188. (This is the default behavior if the option is omitted.) If `false`, the ISCAgent does not start, and the `--ISCAgentPort` option is ignored.

  --ISCAgentPort *NNNN*

Specifies the port to start the ISCAgent on. (The default, if the option is omitted, is 2188.) This option can be used together with `--ISCAgent true`. If the value provided isn't a valid port number (for example, if it is not an integer), or the indicated port is in use, the ISCAgent fails to start.

- Be sure to review Mirroring Communication, Sample Mirroring Architecture and Network Configurations, and Locating the Arbiter to Optimize Mirror Availability to ensure that your deployment addresses all of the needed networking considerations.

5.3 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, the **iris** command 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"]
```

Important: The **iris-main** program confirms that certain Linux capabilities required by the InterSystems IRIS container — for example, CAP_SETUID and CAP_SETGID — before starting InterSystems IRIS. Those that are always required are typically granted by default in container runtime environments; a few that are required only in some circumstances may not be. If one or more capabilities are not present, **iris-main** logs an error and exits without starting the instance. This capability check can be disabled by including the **iris-main** option **--check-caps false**.

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 the instance.

  Important: If the 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 InterSystems IRIS container receives a SIGKILL, on the next start it will engage in normal InterSystems 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 InterSystems 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
  ```

  When a fatal error occurs, **iris-main** directs you to the messages log (see Log Files in the install-dir/mgr Directory) for more information about the error.

  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
-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

Option	Description	Default
-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 is 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. Even when the default password has been changed, the first manual login to each predefined account after the container starts includes a mandatory default password change.	none
--ISCAgent true|false	Starts the ISCAgent on the default ISCAgent port, 2188, on container startup. If false, the ISCAgent does not start, and the **--ISCAgentPort** option is ignored.	true
--ISCAgentPort *NNNN*	Specifies the port to start the ISCAgent on. Can be used together with **--ISCAgent true**.	2188
--check-caps false	Disables automatic Linux capability check before InterSystems IRIS is started.	true
--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 and user-created databases 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, all applications and programs generate and maintain operating and historical data — such as configuration and language settings, user records, and log files — which must be retained beyond the life of a single container to enable upgrades. For this reason, containerization typically must address the need to persist program-related data, including application databases, on durable data storage. This requires a mechanism

that identifies the data to be retained and its persistent location, and can direct the new container to it following an upgrade or a failure of the previous container.

The durable %SYS feature does this for an InterSystems IRIS container by cloning the needed instance-specific data (such as log, journal, and WIJ files and system databases such as **IRISSYS**) to a chosen location on the file system of the container's host and resetting the instance to use the data in the new (cloned) location. To use durable %SYS, you must mount the chosen location as a volume within the container and identify it 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 a mounted external volume, either from within InterSystems IRIS or by using the **iris merge** command to add them when building a custom InterSystems IRIS image from an InterSystems-supplied image, as described in Creating InterSystems IRIS Images, so that they are cloned to durable storage by durable %SYS.

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. (Automating Configuration of InterSystems IRIS with Configuration Merge, Configuration Parameter File Reference)
- The /csp directory, containing the Web Gateway configuration and log files. (Web Gateway Guide)

  Note: Web Gateway containers have a similar durable storage feature, and when this is in use (see Options for Running Web Gateway Containers) the directory is located on its own persistent storage.

- The file /dist/install/misc/buildver, which contains the instance's version, for example 2023.2.0.299.0.
- The file /httpd/httpd.conf, the configuration file for the instance web server. (See Supported Web Servers)
- 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 Custom Items in IRISSYS)
  - The write image journaling file, IRIS.WIJ (which may be relocated to achieve file system separation). (Write Image Journaling and Recovery)
  - The /journal directory containing journal files (which may be relocated to achieve file system separation). (See Journaling; see also Separating File Systems for Containerized InterSystems IRIS)
  - 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.)

    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. For information about accessing the Management Portal of a containerized InterSystems IRIS instance, see Web Access Using the Web Gateway Container.

  - 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)
  - Several InterSystems IRIS system files.

- All databases defined on the instance, beyond the standard InterSystems IRIS databases listed in the first bullet above, that are not read-only. This is to ensure that databases added to the instance in a user-created image based on an InterSystems-supplied image, as described in Creating InterSystems IRIS Images, are included in the durable data. If a database directory is underneath the install directory in the container, for example if it is under /usr/irissys/mgr, it is copied to the corresponding location in the durable %SYS directory. If one or more database directories are not underneath the install directory, a new folder db is created in the install directory and they are copied there.

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).
- Any high availability mechanisms you have in place (*High Availability Guide*) .
- Available storage space and room for expansion (Maintaining Local Databases).

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 a mounted volume, rather than the top level, 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.

Important: Because the instances in the **`iris`** and **`iris-lockeddown`** images were installed and are owned by user **`irisowner`** (UID 51773), as described in Security for InterSystems IRIS Containers, the file system location you specify for durable %SYS *must* first be made writable by **`irisowner`**, for example with this command:

```
chown 51773:51773 root-of-durable-%SYS-directory
```

(You will probably need root privileges to effect this.

When using durable %SYS on Kubernetes *without* the InterSystems Kubernetes Operator, you must include the following security context setting in the pod specification:

```
securityContext:
  fsGroup: 51773
```

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 /volume-path-on-host:/volume-name-in-container
--env ISC_DATA_DIRECTORY=/volume-name-in-container/durable-directory
```

where *volume-path-on-host* is the pathname of the durable storage location to be mounted by the container, *volume-name-in-container* is the name for the mounted volume inside the container, and *durable_directory* is the name of the durable %SYS directory to be created on the mounted volume. For example:

```
docker run --detach
  --volume /data/dur:/dur
  --env ISC_DATA_DIRECTORY=/dur/iconfig
  --name iris21 intersystems/iris:latest-em
```

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

Important: 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: Use the **--publish** option to expose one or more superserver ports on the InterSystems IRIS instance as a port or ports on the host, so that outside entities (including those in other containers) can connect to the instance. The default superserver port is 1972. 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 if necessary.
  - 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.

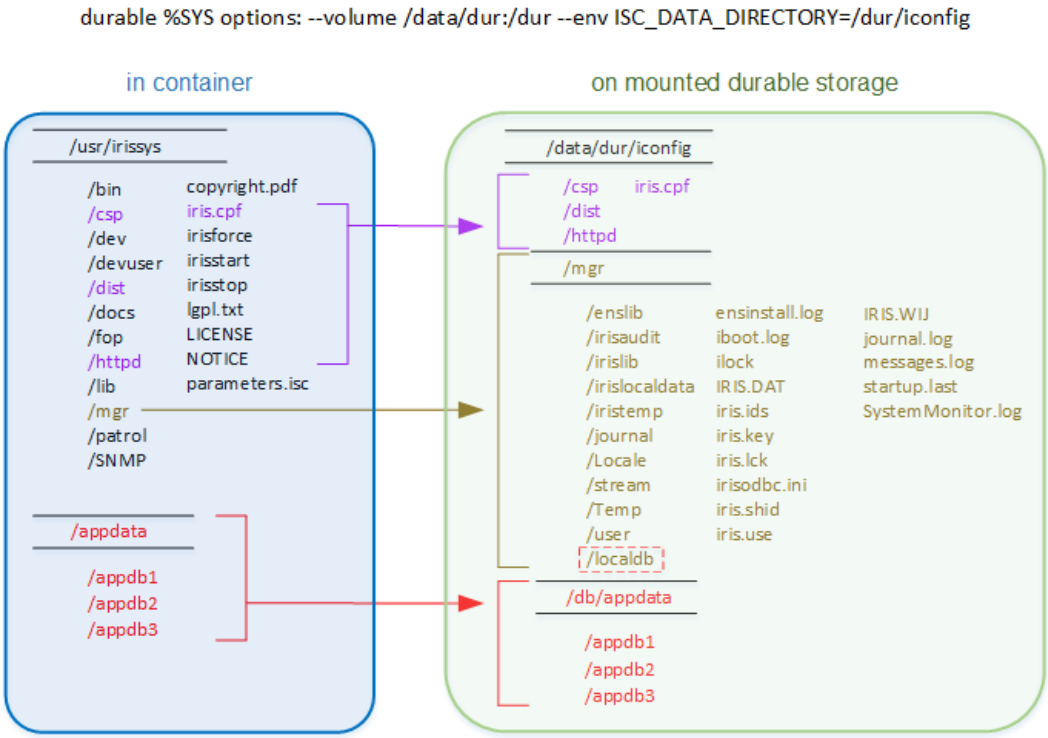
- If the durable %SYS directory specified by the **ISC_DATA_DIRECTORY** environment variable already exists and contains data (file or subdirectories) but does not contain a /mgr subdirectory, no data is copied; the container does not start, and the reason (data other than durable %SYS in the directory) is logged to standard output by **iris-main**, 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.

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

The following illustration shows the relationship between the InterSystems IRIS installation directory and user databases within a container and on the mounted external storage to which they were cloned by durable %SYS (as specified by the options shown). Note that the three application databases outside of the install directory have been cloned to /data/dur/iconfig/db, whereas the **LOCALDB** database, which is within the install directory under /mgr, is cloned to the same location (/data/dur/iconfig/mgr/localdb).

Figure 1: File System Objects Cloned by 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 detailed 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 under "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 cannot operate as an upgrade of the previous 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, the instance's 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.

You can configure separate file systems in deployment using a configuration merge file, as described in Automating Configuration of InterSystems IRIS with Configuration Merge. After the container is started, you can reconfigure the external locations of the primary and secondary directories using the Management Portal (see the following section) 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.

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.

For more information about separation of file systems for InterSystems IRIS, see File System Separation.

5.5 Web Access Using the Web Gateway Container

The InterSystems IRIS Web Gateway provides the communications layer between a hosting web server and InterSystems IRIS for any web application, passing HTTP/HTTPS requests to InterSystems IRIS and returning the responses to be passed to the request originators. The **`webgateway`** image available from InterSystems includes both the Web Gateway and a web server, providing a containerized web server component for stand-alone InterSystems IRIS containers and containerized InterSystems IRIS clusters.

Each InterSystems IRIS instance served by a Web Gateway is identified by a server access profile configured on the Web Gateway; each application to which the Web Gateway sends requests is identified by an application access profile, which

is associated with an application path (such as /applications/appa) configured on the web server. The application access profile determines whether requests for the associated application go to a specific InterSystems IRIS instance only, fail over to another instance if the first is not available, or are load balanced across multiple instances.

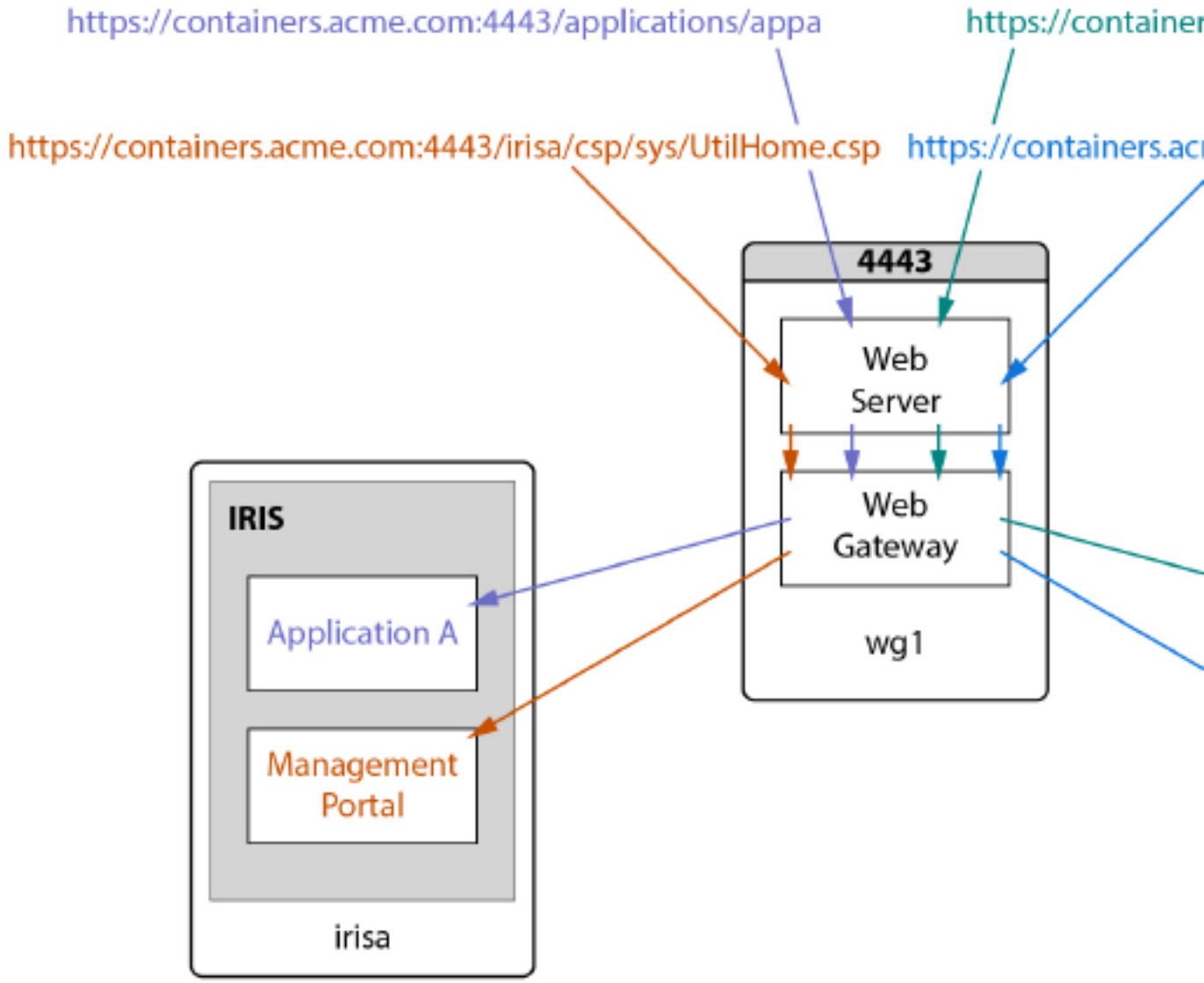
Requests for the Management Portal and other built-in InterSystems IRIS applications, however, always involve the same application path but are intended for a specific InterSystems IRIS instance. For this reason, a different mechanism must be used to route them to the desired instance. For example, if you deploy a containerized distributed cache cluster with a web server tier consisting of Web Gateway containers, each Web Gateway can be configured to load balance application connections across all of the InterSystems IRIS containers serving as application servers, or across a specific subset, but any of the Web Gateway instances must be able to send a Management Portal request to the intended instance.

There are two basic approaches to routing Management Portal and other built-in application requests to specific InterSystems IRIS instances, as follow:

- One-to-many: A single Web Gateway container to distribute both application connections and Management Portal requests to multiple InterSystems IRIS containers.

  This approach uses the instance prefix/CSPConfigName mechanism (described in Structure of an InterSystems Web Application URL) to route requests for the Management Portal and other built-in applications to the desired instance while distributing application connections based on the application path. For example, in the following illustration, a single Web Gateway container, **wg1**, routes requests for /applications/appa to InterSystems IRIS container **irisa** and for /applications/appb to container **irisb** based on their application paths, while routing Management Portal requests prefixed by /irisa to **irisa** and those prefixed by /irisb to **irisb** — the instances in those container having had their CSPConfigName properties set to **irisa** and **irisb**, respectively — even though the application path, /csp/sys, is the same.

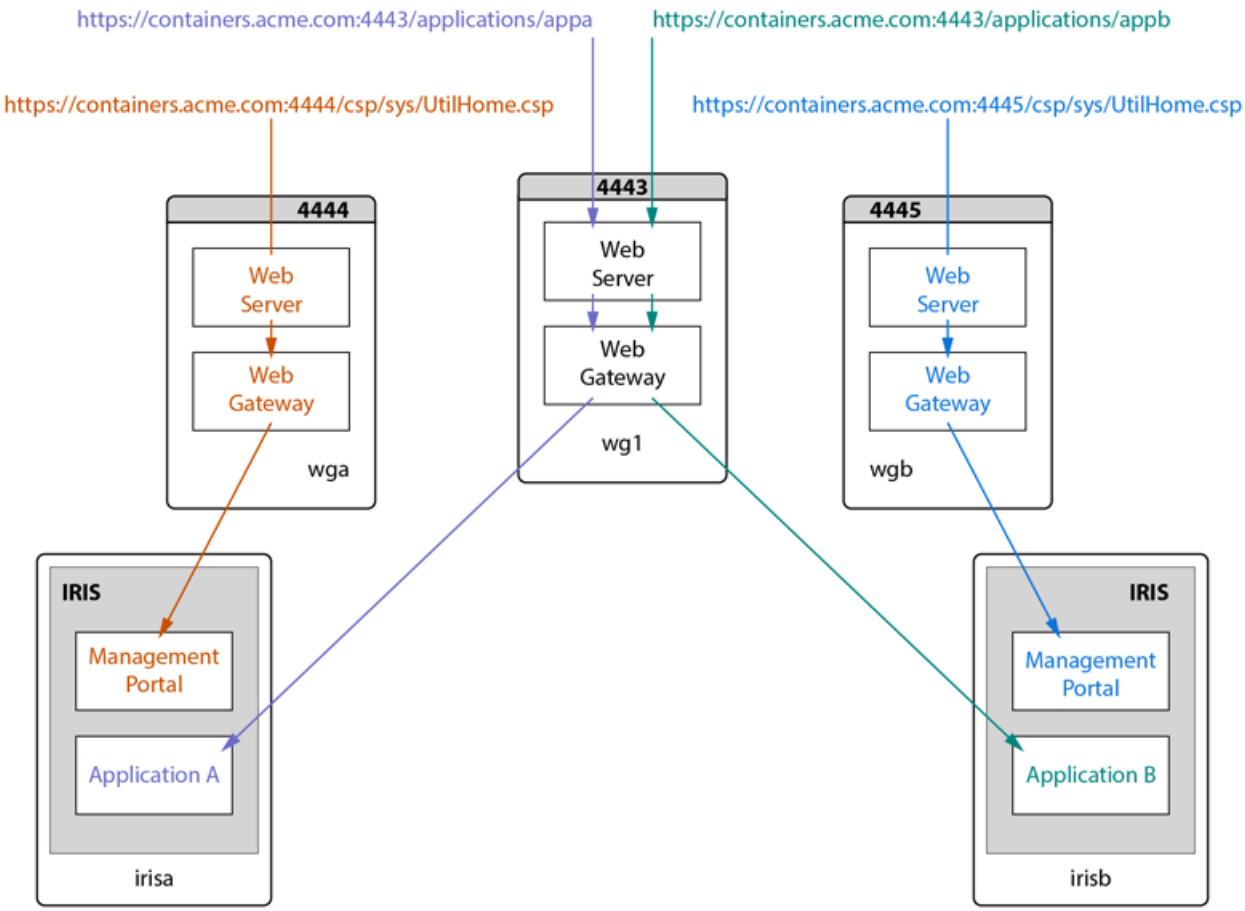
Figure 2: A single Web Gateway container directs application connections and Management Portal requests to multiple InterSystems IRIS containers



- Many-to-many: One or more Web Gateway containers to distribute application connections to multiple InterSystems IRIS containers and a dedicated Web Gateway container for each InterSystems IRIS container to handle Management Portal requests.

  In this approach, one or more Web Gateway containers serving as web server nodes distribute application connections as desired, while Management Portal and other built-in application requests for each InterSystems IRIS instance are handled by a Web Gateway container dedicated to that instance. For example, the following illustration shows one Web Gateway container directing application connections as in the previous illustration, while InterSystems IRIS containers **`irisa`** and **`irisb`** each have a dedicated Web Gateway container directing Management Portal requests to them; the Management Portal URLs are distinguished by the dedicated Web Gateways' published port numbers (rather than URL prefixes based on CSPConfigName values, as in the first approach).

Figure 3: An application Web Gateway container for application requests and dedicated Web Gateway containers for Management Portal requests



Important advantages of the many-to-many approach include:

- Separation of responsibilities.
- Optional customization of permissions and security for different use cases.
- Distribution of Management Portal requests using Docker hostnames assigned to the InterSystems IRIS instances in deployment. This avoids the step of setting each InterSystems IRIS instance's CSPConfigName property (as described above) to a value unique within the configuration, which must be done following deployment, after the instance has started inside the container.

The primary advantage of the one-to-many approach is that it is simpler and requires fewer containers, consuming fewer resources and ports.

A compromise solution might be a two-to-many approach, in which one Web Gateway container serves Management Portal requests as in one-to-many, and another Web Gateway distributes application connections across the InterSystems IRIS instances.

You can find examples of both approaches, including all the needed files and some useful scripts, at https://github.com/intersystems-community/webgateway-examples.

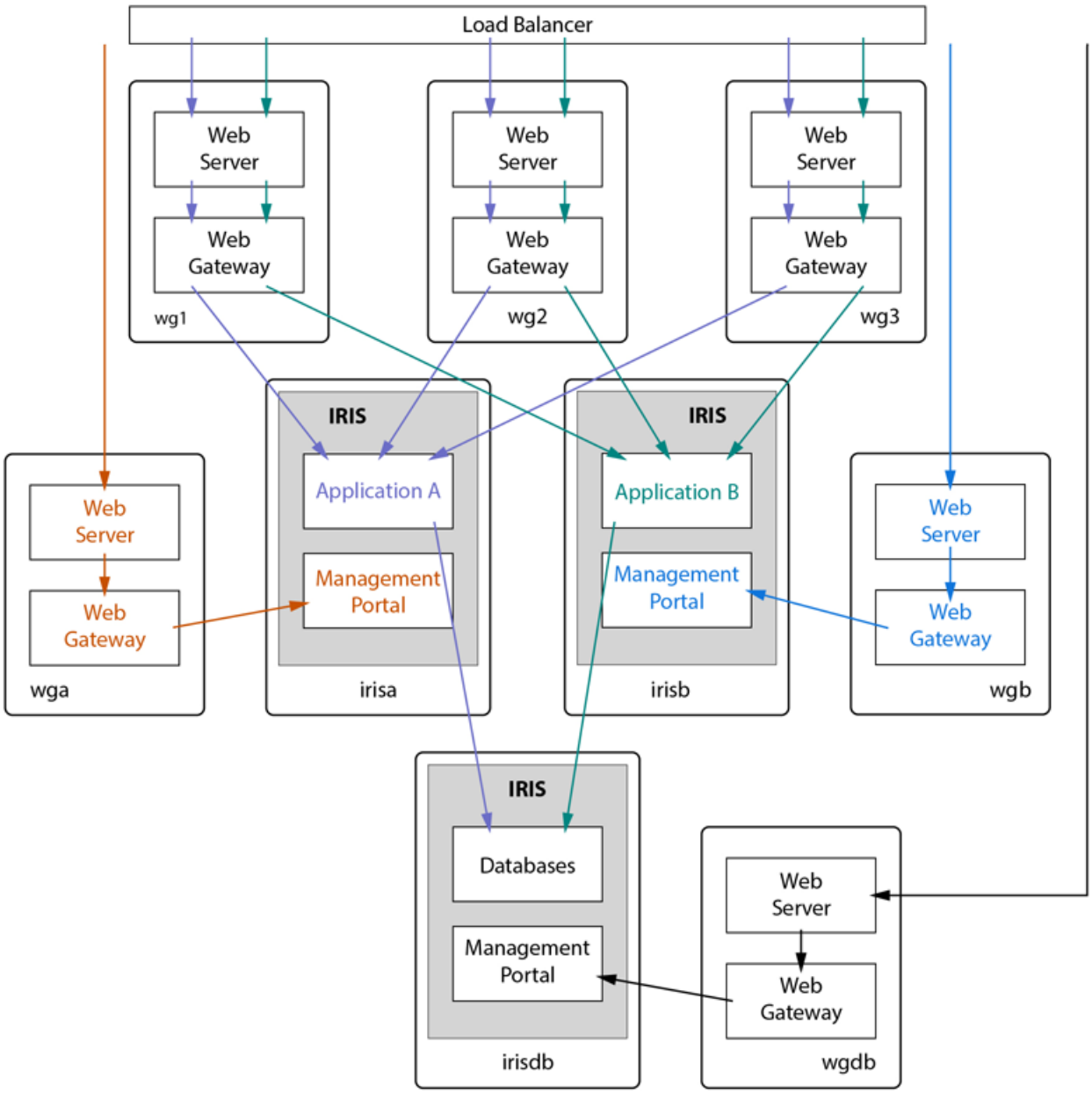
Note: The IKO (see Automated Deployment of InterSystems IRIS Containers) deploys dedicated Web Gateway containers for Management Portal access by including one as a sidecar container in each InterSystems IRIS pod; for more information, see Using the InterSystems Kubernetes Operator.

The Web Gateway approach you take depends on the containerized configuration involved, the expected amount of application traffic, the needs of your organization, and other factors. Generally speaking, the one-to-many approach works well for less complex configurations and smaller organizations seeking simplicity and a minimal footprint. For larger enterprises and more complex configurations with a focus on stability, resiliency, and availability and a need to separate instance-local connections from the primary application traffic, the many-to-many approach may be preferable. As noted above, there are also various possible combinations; for example, when using the one-to-many approach, you may want to include a dedicated Web Gateway for a particular InterSystems IRIS instance that handles heavy traffic at peak to avoid increasing latency for the others.

When Web Gateway nodes are used in a web server tier distributing application connections across multiple instances, the manner of distribution can depend on the configuration. For example:

- The general best practice for sharded clusters is to distribute application connections equally across all data nodes (and compute nodes if present) in the cluster, so in this case you would configure the application profile(s) to distribute all application connections equally across all of the data node containers defined in the server profiles. Because Management Portal connections to a sharded cluster are typically made only to node 1, however, these can easily be routed by the web server tier without the need for any additional Web Gateway nodes.
- In a distributed cache cluster, on the other hand, you might want to maximize caching efficiency by directing requests for different applications to different application servers; in such a case, the application paths would be configured to distribute connections differentially. When this is done and large web server and application server tiers are involved, the use of dedicated Web Gateways may be optimal. This is shown in the following simplified illustration, in which the InterSystems IRIS containers (two application servers and one data server) each has a dedicated Web Gateway container to handle Management Portal and other built-in application requests received over networks separate from the one conveying application requests to the load balancer.

Figure 4: Containerized distributed cache cluster with application connections directed differentially and dedicated Web Gateways



Note: For an important discussion of load balancing a web server tier distributing application connections across multiple InterSystems IRIS instances, see Load Balancing, Failover, and Mirrored Configurations.

Important: In versions of InterSystems IRIS prior to 2023.2, the Web Gateway and a preconfigured private web server were installed with InterSystems IRIS by default, including in containers. For this reason, if you are upgrading from a pre-2023.2 version to the current version, you must update all deployment scripts and tools to reflect the new deployment options described by this document. Further, 2023.2 and later InterSystems IRIS containers should be used only with InterSytems Kubernetes Operator 3.6 and later.

For the convenience of those testing and evaluating InterSystems IRIS, the InterSystems IRIS Community Edition image continues to include the Web Gateway and preconfigured web server; the web server can be reached (to access the Management Portal, for example) at whatever host port is published for the containerized instance's web server port, 52773.

5.5.1 InterSystems Web Gateway Images

There are three types of **webgateway** images available from InterSystems, each of which provides a web server component for containerized deployments:

- The **webgateway** image contains the following components, installed by root in the indicated locations within the container:
  - An InterSystems Web Gateway in /opt/webgateway.
  - An Apache web server in /etc/apache2.
- The **webgateway-nginx** image contains the following components, installed by root:
  - An InterSystems Web Gateway in /opt/webgateway.
  - An Nginx web server in /opt/nginx.
- The **webgateway-lockeddown** image, designed to meet the strictest security requirements, contains the following nonroot components installed, owned, and run by **irisowner** (UID 51773) :
  - An InterSystems Web Gateway installed in /home/irisowner/webgateway with locked-down security.
  - An Apache web server installed in /home/irisowner/apache and configured to use port 52773 instead of the standard port 80.

Note: For details about nonroot installation and locked-down security, see Security for InterSystems IRIS Containers.

5.5.2 Configuring the Web Gateway

The Web Gateway is configured using the Web Gateway management pages, but the configuration is contained in the CSP.ini file (much as an InterSystems IRIS instance's configuration is contained in the iris.cpf file). The Web Gateway is installed with a minimal default version of the CSP.ini file containing some basic settings. In Web Gateway containers only, the CSP.conf file, which configures the web server to interact with the Web Gateway, is added to the web server's configuration. For more information about the contents of the CSP.ini file, see Web Gateway Configuration File (CSP.ini) Parameter Reference; for information about the contents of the CSP.conf files provided in Web Gateway containers, see Recommended Option: Apache API Module without NSD and Using the NSD with Nginx.

All three **webgateway** images contain minimal default versions of both the CSP.ini file and an Apache or Nginx-specific CSP.conf file (in the appropriate directory as indicated in the preceding list). However, you can provide your own version of one or both files to overwrite these. There are multiple ways to prepare your initial version of the CSP.ini file. For example, you might modify a file from an existing or previous Web Gateway deployment and use that for one or more Web Gateway containers, or perhaps start in one container with either your own file or the basic default version provided, use the management pages to modify the configuration as needed, then copy the updated file and use it with additional containers. The Web Gateway Guide provides comprehensive information on Web Gateway configuration.

A Web Gateway container can optionally be run with a version of the durable %SYS feature, which creates a durable data directory called webgateway within the container in which the Web Gateway's configuration is stored (see Options for Running Web Gateway Containers). If you use this option, the CSP.ini and CSP.conf files (default or user-provided) are copied to that directory, as is the CSP.log log file; this enables you to upgrade the container without losing the existing configuration, and to simultaneously update multiple Web Gateway configurations, as described in the following section.

5.5.3 Synchronized Reconfiguration of Multiple Web Gateway Containers

In the case of a dedicated Web Gateway container, the Web Gateway's configuration is relatively simple, including as it does a single server access profile for the InterSystems IRIS instance it is dedicated to and application access profiles for

only the applications that instance is serving. Therefore reconfiguring a dedicated Web Gateway can be as simple as accessing its management pages and making the needed changes.

However, when you deploy multiple Web Gateway containers as part of the same cluster, as in a web server tier and for shared Management Portal access, you will likely need a method for simultaneously updating their configurations, in particular their server access profiles and application access profiles, for example when you add an InterSystems IRIS node or change an application path. The CSP.ini merge feature provides a convenient means of doing this. If you provide your own CSP.ini file at container creation, staged on a mounted external volume as described in Options for Running Web Gateway Containers, it is copied once to either the container's file system or the durable data directory if it exists. Following that, if the staged file remains in its original location and accessible to the Web Gateway containers, updates to the file are automatically merged to each Web Gateway instance's active CSP.ini file (in either location), along with the setting `[SYSTEM]/RELOAD=1`, which causes the configuration to be reloaded by the Web Gateway within a minute. Therefore, if you deploy multiple Web Gateway containers by mounting the same staged CSP.ini file on the same external volume for all of them, you can reconfigure them all identically and simultaneously by updating the staged file. This approach works equally well for reconfiguring a dedicated Web Gateway, a single Web Gateway distributing application connections to multiple InterSystems IRIS containers, or a shared Web Gateway for management portal access.

When using this merge update approach, whatever the number of Web Gateway containers involved, it is important to takes steps to avoid changing the instances' CSP.ini files by other means (such as through the Web Gateway management pages) or, if you do make such changes, do one of the following:

- Remove fields from the staged CSP.ini file that you have changed by other means, possibly by deleting everything other than the fields you want to update.
- Apply updates you have made by other means to the staged file. For example, you might change the file using the management pages, then copy the modified file to the staging location, overwriting the original staged file. If you are using the staged file for multiple containers, this would allow you to test the changes you want to make on one instance and then apply them to the rest through a merge update.

Note: The staged CSP.ini file is never merged unless it has been modified. When the durable data directory is in use, a copy of the last merged file is kept in */webgateway/CSP.ini.last_merge* in the durable %SYS directory.

Externally staging your own CSP.conf file allows you to keep the web servers in multiple Web Gateway contains updated in the same fashion as described for CSP.ini, above. As long as the file remains in its original staged location and accessible to the Web Gateway containers, it is monitored by each Web Gateway container for changes, and when these are detected it is recopied to the appropriate directory and the web server reloads the configuration and restarts. This is especially helpful when you update the Web Gateway server access profiles in the staged CSP.ini and need to update the application paths configured on the web servers accordingly.

5.5.4 Web Gateway Container Security

Securing a Web Gateway instance involves two primary tasks:

- Securing the Web Gateway's connections to the InterSystems IRIS instances configured in its server access profiles, which entails, at a minimum, requiring authentication to InterSystems IRIS using an existing user account on the target instance. However, InterSystems strongly recommends adding TLS encryption. This topic is covered in detail in Protecting Web Gateway Connections to InterSystems IRIS and Configuring the Web Gateway to Connect to InterSystems IRIS Using TLS.

  Note: InterSystems also strongly recommends securing connections between web clients and the web server with TLS.

- Securing access to the Web Gateway's management pages, which involves requiring authentication to the Web Gateway, as well as limiting the hosts from which the management pages can be accessed. For information on this topic, see

Overview of the Web Gateway Management Pages and "Security" in Configuring the Default Parameters for the Web Gateway.

In the default CSP.ini in the Web Gateway container (and by default in locally installed Web Gateways), the **CSPSystem** predefined user account is used for both purposes. You can, however, use any credentials you want for either purpose. When securing connections to InterSystems IRIS instances, the credentials you configure in each server access profile must be valid on that particular instance; otherwise there are no restrictions on either set of credentials, and using a purpose-made account rather than **CSPSystem** allows you to more closely restrict information about the credentials.

When deploying a clustered configuration involving multiple Web Gateways interacting with multiple InterSystems IRIS instances, as discussed in Synchronized Reconfiguration of Multiple Web Gateway Containers, using one set of credentials for all connections to InterSystems IRIS and another set for management access to all Web Gateways is more convenient (although using different sets enhances security by reducing credentials reuse). A reasonably convenient and (if other precautions are followed) reasonably secure approach to implementing this is as follows:

- Deploy the InterSystems IRIS containers using configuration merge `[Actions]` parameters (as described in Create, Modify and Delete Security Objects in *Automating Configuration of InterSystems IRIS with Configuration Merge*) to create an account specifically for Web Gateway authentication, with the password encrypted using the `PasswordHash` parameter). Stage the merge file on a mounted external volume, as required for continuous monitoring and merging.
- Deploy the Web Gateway containers with a custom CSP.ini file that specifies:
  - In the server access profiles, the Web Gateway access credentials created during InterSystems IRIS deployment.
  - A different set of credentials for management pages access.
  - A restricted set of IP addresses from which the management pages can be accessed.

  Stage the CSP.ini file on a mounted external volumes, as required for continuous monitoring and merging.

Continuous monitoring and merging of the CPF merge and CSP.ini files enables you to use the CSP.ini merge feature for later synchronized reconfiguration of the Web Gateway containers, including the security best practice of regular password changes for both the server access credentials (in both the InterSystems IRIS containers and Web Gateway containers) and the Web Gateway management credentials, or even regularly creating a new account on the InterSystems IRIS instances and updating the CSP.ini accordingly.

Important: When deploying one or more Web Gateway containers with a custom CSP.ini file, three passwords in the file should always be encrypted before deployment:

- `[SYSTEM]/Password` — The Web Gateway management access password.
- `[<server>]/Password` — The InterSystems IRIS authentication password (one in each server access profile).
- `[<server>]/SSLCC_Private_Key_Password` — The TLS private key password, if TLS is in use (one in each server access profile).

The Web Gateway management access password, `[SYSTEM]/Password`, must be encrypted before you enter it into the CSP.ini file. For the two passwords in each server access profile, you have two options, as follows:

1. Deploy a Web Gateway container with the CSP.ini you want to use.
2. For `[<server>]/Password`, `[<server>]/SSLCC_Private_Key_Password`, or both, enter one of the following:
   - A plain text password. When the Web Gateway starts up, or when it is reloaded while running by the addition of the setting `[SYSTEM]/RELOAD=1` to the CSP.ini file, the password is automatically encrypted. (When using scripts and other automated methods, you can use the **CSPpwd** utility to encrypt all plain text passwords in a CSP.ini file, but note the restrictions on password decryption on Windows platforms.)
   - On UNIX® and Linux systems, a command enclosed in braces, for example `Password={sh /tmp/PWretrieve.sh}`. When the Web Gateway starts up or is reloaded, the command is executed and the result is stored in memory only as the value of the field. This allows you to retrieve passwords from sources such as cloud platform or third-party secret managers without ever committing them in plain text to durable storage.

As described in Authentication and Passwords, you must change the default password for the predefined accounts, including **`CSPSystem`**, on any InterSystems IRIS instance as part of deployment or immediately after. If you do choose to make use of **`CSPSystem`** or one of the other predefined accounts in your CSP.ini file, be sure to provide yourself with secure access to the encrypted post-deployment password so that you can accurately add it to the CSP.ini file.

Note: When securing connections between a containerized Web Gateway and InterSystems IRIS instances with TLS (as described in Protecting Web Gateway Connections to InterSystems IRIS and Configuring the Web Gateway to Connect to InterSystems IRIS Using TLS) or using SSL mode to secure the Apache web server, the best practice for providing a certificate is to generate a passwordless server key and mount both the key and the certificate as Docker secrets (or Kubernetes secrets if applicable). When you need to update the certificate, you can simply update the secrets. You can also create a server key with password and mount the password as a separate secret from the key and certificate. This practice avoids having to manually provide a server key password, which compromises resiliency, or recording the password so it can be automatically retrieved, which compromises security.

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
- Options for running Web Gateway containers

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 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.

The image tags in the examples in this document, for example `2023.2.0.299.0` in the following, may be out of date. Before attempting to download an image, consult Using the InterSystems Container Registry for the current image tags.

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. The example shown below does the following:
  - Publishes the instance's default superserver port (1972) to port 9092 on the host.
  - Includes the needed options for durable %SYS (see Ensuring that Durable %SYS is Specified and Mounted).
  - Uses the **iris-main -key** option, in which 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 --detach --publish 9092:1972
  --volume /data/durable:/dur
  --env ISC_DATA_DIRECTORY=/dur/iris_conf.d
  intersystems/iris:latest-em --key /dur/key/iris.key
```

- This example adds a configuration merge file staged on the durable data volume, containing settings to be merged into the InterSystems IRIS instance's CPF (see Automated Deployment of InterSystems IRIS Containers) 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 --detach --publish 9092:1972
  --volume /data/durable:/dur
  --env ISC_DATA_DIRECTORY=/dur/iris_conf.d
  --env ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf intersystems/iris:latest-em
  --key /dur/key/iris.key
```

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

Note: Because the InterSystems IRIS Community Edition image described in Downloading the InterSystems IRIS Image includes a free temporary license, the **--key** option 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 InterSystems IRIS image testing

# Definitions to toggle_____________________________________________
container_image="intersystems/iris:latest-em"

# the docker run command
docker run -d
  --name iris
  --hostname iris
# expose default superserver port
  -p 9091:1972
# mount durable %SYS volume
  -v /data/durable:/dur
# specify durable %SYS directory and CPF merge file
  --env ISC_DATA_DIRECTORY=/dur/iris_conf.d
  --env ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf
# enable allocation of huge pages
  --cap-add IPC_LOCK
  $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:latest-em
    command:  --key /dur/key/iris.key
    hostname: iris
    ports:
    # the default superserver port
    - "9091:1972"
    volumes:
    # the durable %SYS volume
    - /data/durable:/dur
    environment:
    # the durable %SYS directory
    - ISC_DATA_DIRECTORY=/dur/iris_conf.d
    # the CPF merge file
    - ISC_CPF_MERGE_FILE=/dur/merge/merge.cpf
```

5.6.4 Options for Running Web Gateway Containers

In production, the recommended method for deploying multicontainer InterSystems IRIS clusters including multiple Web Gateway containers is the InterSystems Kubernetes Operator (IKO), described in Automated Deployment of InterSystems IRIS Containers. Version 3.6 of the IKO deploys dedicated Web Gateway containers for Management Portal access (the many-to-many approach) by including one as a sidecar container in each InterSystems IRIS pod; for more information, see Using the InterSystems Kubernetes Operator.

For development and testing purposes, there are three basic methods for deploying InterSystems IRIS containers together with Web Gateway containers, as follows:

- Docker Compose and other scripting tools

- Kubernetes
- A user-created InterSystems IRIS image containing a Web Gateway instance and a web server in addition to the InterSystems IRIS instance. (For a general discussion of user-created InterSystems IRIS images, see Creating InterSystems IRIS Images.)

You can find examples of these methods, including all the needed files and some useful scripts, at https://github.com/intersystems-community/webgateway-examples/tree/master. The remainder of this section discusses only the options available when running Web Gateway containers on the command line.

The only required option when creating and starting a Web Gateway container is publishing container ports 80 and 443 to host ports so that other entities can contact the Web Gateway and the web server, as in the following command:

```
docker run -d --publish 8080:80 --publish 4443:443
  containers.intersystems.com/intersystems/webgateway:latest-em
```

The following are optional:

- Durable data directory — To create a durable data directory called webgateway within the container, in which the Web Gateway's configuration files are stored, as discussed in Configuring the Web Gateway, use the **--volume** option to mount a persistent data volume and the **ISC_DATA_DIRECTORY** environment variable to specify a location on it for the directory. For example, the following command would create a durable data directory at /dur/webgateway inside the container and at/nethome/user21/dur/webgateway on the host's file system.

  ```
  docker run -d --name wg11 --publish 80:80 --publish 443:443
    --volume /nethome/user21/dur:/dur --env ISC_DATA_DIRECTORY=/dur
     containers.intersystems.com/intersystems/webgateway:latest-em
  ```

  This is equivalent to the procedure for enabling the durable %SYS feature for InterSystems IRIS containers; for detailed information about using these options and durable %SYS generally, see Durable %SYS for Persistent Instance Data.

  Important: Because the **webgateway-lockeddown** image contains a Web Gateway instance and web server installed and owned by user **irisowner** (UID 51773), the host file system location you specify for the durable data directory of a locked down container *must* be writable by **irisowner**. (You will most likely need root privileges to effect this.)

  When you run a **webgateway** container with the durable data option, the following occurs:

  – The specified external volume is mounted.

  – If the webgateway directory does not exist in the location specified by **ISC_DATA_DIRECTORY**, it is created and the configuration files are copied there for use by the Web Gateway, as follows:

    - Configuration files you provide are copied to the webgateway directory, with links to their default locations in the container.
    - Configuration files you do not provide are copied from the default locations within the container, with links to those locations.

  – If the webgateway directory already exists in the location specified by **ISC_DATA_DIRECTORY**, it is assumed to be the data directory for a previous **webgateway** container, and

    - If it contains the expected Web Gateway configuration files, these are linked to their locations in the container and are used by the Web Gateway; options specifying user-provided configuration files (as described below) are ignored.

Important: Because you cannot provide your own configuration files (as described below) when deploying a **webgateway** container using an existing durable webgateway directory, you cannot upgrade and reconfigure a containerized Web Gateway in the same operation. Instead, start by deploying a new version of the container using the previous container's durable webgateway directory, then reconfigure the Web Gateway as needed.

- If one or more of the expected Web Gateway configuration files are not present, the webgateway is assumed to be corrupted; an error is logged and the container fails to start.

Important: When upgrading a **webgateway** container that uses a durable data directory to version 2021.2 or later using a **webgateway-lockeddown** image, you *must* make the existing durable directory writable by user **irisowner**.

The default, if you do not use the **ISC_DATA_DIRECTORY** variable to specify a location on writeable persistent storage accessible to the Web Gateway within the container, is to not create a durable data directory and maintain the configuration files in their default locations (as previously described).

- User-defined configuration files — To provide your own CSP.ini file, use the **--volume** option to mount a persistent data volume (separate from the durable data directory volume specified by **ISC_DATA_DIRECTORY** (if in use), stage the file on that volume, and use the **ISC_CSP_INI_FILE** environment variable to indicate its location. If you also create a durable webgateway directory, as described in the preceding, the file is copied to that directory and linked to its original location within the container; if not, it is copied to the original location, overwriting the default file.

  Important: If you want to use the merge update approach, as described in Configuring the Web Gateway, the staged CSP.ini file must remain in place and accessible to the container.

  To supply your own CSP.conf file, do the same and use the **ISC_CSP_CONF_FILE** environment variable to specify its location. For example, to create a durable webgateway directory and provide your own CSP.ini and CSP.conf files to be used for the Web Gateway's configuration, you would use a command like the following:

```
docker run -d --name wg11 --publish 80:80 --publish 443:443
  --volume /nethome/user21/dur:/dur --env ISC_DATA_DIRECTORY=/dur
  --volume /nethome/user21/config:/config --env ISC_CSP_INI_FILE=/config/CSP.ini
  --env ISC_CSP_CONF_FILE=/config/CSP.conf
  containers.intersystems.com/intersystems/webgateway:latest-em
```

  The default, if one of these variable is not specified, is to use the basic default file located within the container.

- To enable the Apache web server's SSL module, add the **--ssl** entrypoint option (following the image specification). The default, if the option is omitted, is not to enable the Apache SSL module.
- To identify the Web Gateway's server to Apache, use the **--server** *server-name* entrypoint option. The default, if the option is omitted, is to use the container's ID (unless the Docker **--hostname** option is included, in which case the value provided is used).

The following is an example of a **docker run** command using all the options described:

```
docker run -d --name wg11 --publish 80:80 --publish 443:443
  --volume /nethome/user21/dur:/dur --env ISC_DATA_DIRECTORY=/dur
  --env ISC_CSP_INI_FILE=/dur/CSP.ini --env ISC_CSP_CONF_FILE=/dur/CSP.conf
  containers.intersystems.com/intersystems/webgateway:latest-em
  --ssl --server webgateway11
```

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.

Important: Before upgrading an InterSystems IRIS container that uses durable %SYS to version 2021.2 or later, you *must* make the existing durable directory writable by user 51773, for example with this command.

```
chown -R 51773:51773 root-of-durable-%SYS-directory
```

You will most likely need root privileges to make this change,

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 --detach --publish 9091:1972
  --volume /data/durable:/dur --env ISC_DATA_DIRECTORY=/dur/iconfig
  intersystems/iris:<version_number> --key /dur/key/iris.key
```

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 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.

Note: For information about upgrading a Web Gateway container with a durable data directory, see Options for Running Web Gateway Containers.

5.8 Creating InterSystems IRIS Images

For most use cases, the simplest and least error-prone approach to creating a custom Docker image for InterSystems IRIS is to base the Dockerfile on an existing image from InterSystems, in which InterSystems IRIS is already installed with the **iris-main** program as the entrypoint. You can then add the dependencies relevant to your own application solution and start the InterSystems IRIS instance, optionally use the configuration merge feature to modify its configuration, and issue other commands to it.

For example, suppose you want to create an InterSystems IRIS image that includes your application and the two predefined databases it requires. You can create a Dockerfile to do the following (as illustrated in the example following the steps):

1. Start with an InterSystems IRIS image as base.
2. Switch to user **root** and upgrade the base's third-party dependencies.

   Important: Upgrading the packages in the base is a best practice that helps avoid security vulnerabilities.

3. Switch back to the instance owner user, **irisuser**/**51773** (see Ownership and Directories).
4. Copy in a file containing the application code.
5. Copy in a configuration merge file to create the needed namespaces and application databases on the instance, change the default password using the PasswordHash parameter as described in Authentication and Passwords, and make any other desired configuration changes. Such a merge file might look like this:

   ```
   [Startup]
   PasswordHash=dd0874dc346d23679ed1b49dd9f48baae82b9062,10000,SHA512
   [Actions]
   CreateDatabase:Name=DB-A,Directory=/usr/irissys/mgr/DB=A,Size=5368,MaxSize=536871,ResourceName=%DB_%DB-A
   CreateDatabase:Name=DB-B,Directory=/usr/irissys/mgr/DB=B,Size=5368,MaxSize=536871,ResourceName=%DB_%DB-B
   CreateNamespace:Name=DB-A,Globals=DB-A,Routines=SYS
   CreateNamespace:Name=DB-B,Globals=DB-B,Routines=SYS
   ```

6. Start the InterSystems IRIS instance.
7. Execute the **iris merge** command with the configuration merge file to reconfigure the instance, including creating the databases.
8. Shut down the instance.
9. Copy in prepared IRIS.DAT database files, overwriting those created in the previous step.
10. Remove the merge file, unless you plan to specify it as the instance's merge file going forwardusing the **ISC_CPF_MERGE_FILE**

The following sample Dockerfile illustrates the above steps:

```
FROM intersystems/iris:latest-em

USER root

RUN apt-get update && apt-get -y upgrade \
 && apt-get -y install unattended-upgrades

USER 51773

COPY application.xml /

COPY merge.cpf /tmp/

RUN iris start IRIS \
 && iris merge IRIS /tmp/merge.cpf \
 && iris stop IRIS quietly
```

```
COPY DB-A.DAT /usr/irissys/mgr/DB-A
COPY DB-B.DAT /usr/irissys/mgr/DB-B

RUN rm /tmp/merge.cpf
```

Important: Before attempting to build your own InterSystems IRIS image as described in this section, be sure to familiarize yourself with the information in the three sections covering the **iris-main** program, the durable %SYS feature, and the containerization tools provided with InterSystems IRIS.

Note: 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.

To clone user-defined databases like those included in the example to the durable data location created by durable %SYS, the database files you copy in must be writable.

If for any reason you want to remove the contents of the messages.log or console.log files after InterSystems IRIS is installed, so that when the instance starts in the container one or both of these files is empty, do not delete the file(s), because the **iris-main** program treats a missing log file as a fatal error. Instead, you can empty them, reducing their size to zero.

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.

For an example of using this approach to create an InterSystems IRIS image that includes a Web Gateway instance and a web server, go to https://github.com/intersystems-community/webgateway-examples/tree/master/demo-dockerfile

5.9 InterSystems IRIS Containerization Tools

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

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

5.9.1 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. If you install InterSystems IRIS instance from a kit in your Dockerfile, rather than using an InterSystems image as a base as described in Creating InterSystems IRIS 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.	irisowner
ISC_PACKAGE_IRISGROUP	Effective user for InterSystems IRIS processes.	irisowner
ISC_PACKAGE_MGRUSER	Username of the installation owner.	irisowner
ISC_PACKAGE_MGRGROUP	Group that has permission to start and stop the instance.	irisowner

Note: If you are building your own InterSystems IRIS image, you can optionally 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.

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

5.9.2 SYS.Container API

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 InterSystems 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 **iris-main --password-file** option and the password change script, as described in Authentication and Passwords.

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

  Changes the Web Gateway management password (see Overview of the Web Gateway Management Pages); called by the **iris-main --password-file** option and the password change script, as described in Authentication and Passwords.

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

  Sets **Change password on next login** on user enabled accounts with at least one role (see User Account Properties).

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

  Disables password-based login for a specified user; other forms of authentication (see Authentication: Establishing Identity) remain enabled.

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

  Enables OS-based authentication for the instance (see About Operating-System–Based Authentication).

- **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).

- **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 Using System Monitor.

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

A common approach is to include these methods in a Dockerfile based on an InterSystems IRIS image from InterSystems (as described in Creating InterSystems IRIS Images) by calling them through the **iris terminal** command at instance startup, for example:

```
RUN iris start $ISC_PACKAGE_INSTANCENAME \
    && iris terminal $ISC_PACKAGE_INSTANCENAME -U %SYS
"##class(SYS.Container).PreventJournalRolloverMessage()"
    && iris terminal $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Container).SetMonitorStateOK()"
    && iris terminal $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Container).QuiesceForBundling()"
    && iris terminal $ISC_PACKAGE_INSTANCENAME quietly
```

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.

6.1 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.

6.2 Accessing Endpoints Elsewhere on the Host from Within a Container

The applications that you deploy on your containerized InterSystems IRIS instance may need to communicate with network endpoints which are hosted elsewhere on the container's host machine. (For example: perhaps you are running an OAuth 2.0 server in another container.)

Your containerized application cannot access any endpoints outside the container using the hostname `localhost`, because the container serves as an isolated environment. For information about the mechanisms that Docker provides for networking its containers, refer to the following Docker documentation pages:

- Docker: https://docs.docker.com/desktop/networking/#networking-features
- Docker Compose: https://docs.docker.com/compose/networking/

6.3 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.