Automating Configuration of InterSystems IRIS with Configuration Merge

Version 2023.3
2024-05-16

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

Automating Configuration of InterSystems IRIS with Configuration Merge
InterSystems IRIS Data Platform Version 2023.3 2024-05-16

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

Automating Configuration of InterSystems IRIS with Configuration Merge

This document explains how to use configuration merge to deploy or reconfigure InterSystems IRIS® data platform.

1 What is configuration merge?

The configuration merge feature lets you make as many changes as you wish to the configuration of any InterSystems IRIS instance in a single operation. To use it, you simply record the changes you want to make in a declarative configuration merge file and apply that file to the instance, when it is deployed or at any later point. Configuration merge can easily be used to automatically deploy multiple instances with varying configurations from the same container image or kit, as well as to simultaneously reconfigure multiple running instances, enabling automated reconfiguration of clusters or other multi-instance deployments. Configuration merge can be used in deployment of any InterSystems IRIS instance, containerized or locally installed, on any supported UNIX® or Linux platform, including Linux cloud nodes. You can reconfigure running instances using configuration merge on both UNIX/Linux and Windows platforms.

For examples of configuration merge files used to deploy containers, see Useful Parameters in Automated Deployment. The *Configuration Parameter File Reference* contains a comprehensive description of all InterSystems IRIS configuration parameters.

2 How is InterSystems IRIS configured?

The configuration of an InterSystems IRIS instance is determined by a file in its installation directory named iris.cpf, which contains configuration parameters as name/value pairs. Every time the instance starts, including for the first time after it is deployed, it reads this *configuration parameter file*, or CPF, to obtain the values for these settings. This allows you to reconfigure an instance at any time by modifying its CPF and then restarting it.

For example, the globals setting in the `[config]` section of the CPF determines the size of the instance's database cache. The setting in the CPF of a newly installed instance specifies an initial cache size equal to 25% of total system memory, which is not intended for production use. To change the size of the database cache, you can open the instance's CPF in any text editor and specify the desired cache size as the value of globals, then restart the instance. Most parameters can be changed using other methods; for example, you can also modify the value of globals using the Management Portal or using the methods in the persistent class Config.config. Updating an instance's CPF, however, is the only general mechanism that lets you make multiple configuration changes to an instance in a single operation and automate the simultaneous configuration of multiple instances.

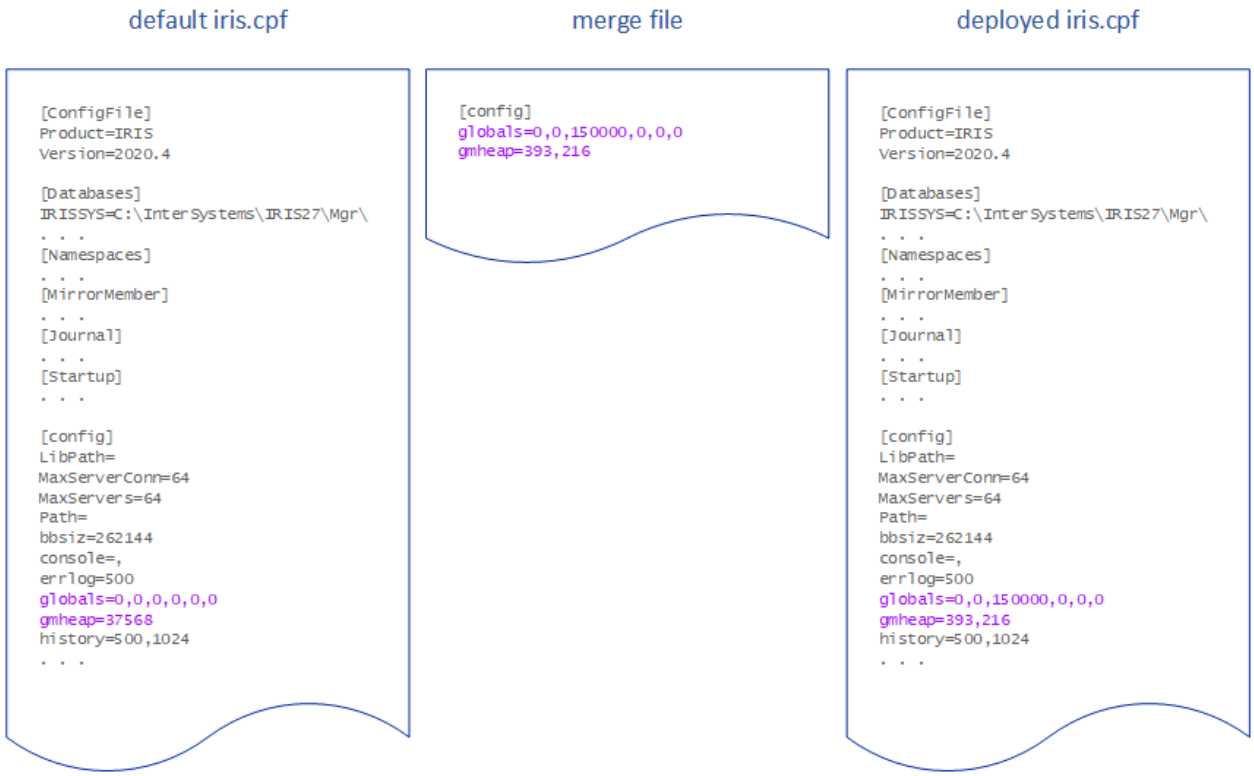
For an overview of the use and contents of the CPF, see the *Configuration Parameter File Reference*.

3 How does configuration merge work?

A *configuration merge file* is a partial CPF that contains any desired subset of InterSystems IRIS configuration parameters and values. When a merge file is applied to an instance with configuration merge, those settings are merged into the instance's CPF, replacing the values, as if you had edited the CPF and changed the values manually. If a parameter in the merge file is not present in the original CPF, it is simply added in the appropriate place.

For example, the data and compute nodes in a sharded cluster typically require with a database cache that is much larger than that generally configured for other purposes, and have more shared memory configured as well. To configure an instance to have a larger database cache and more shared memory when deployed, or to reconfigure an existing instance this way, you can apply a configuration merge file that includes the globals parameter (which specifies the size of the database cache) and the gmheap parameter (which specifies the amount of shared memory) with the desired values; these replace the default values in the CPF of the instance. The following illustrates the use of a merge file to update both the parameters when deploying an instance:.

Figure 1: Merge File Updates Memory Settings During Deployment



4 Can configuration merge customize more than the configuration?

In addition to changing the values of configuration parameters, configuration merge can create, modify, and delete dozens of different InterSystems IRIS objects, such as namespaces and databases; users, roles, and resources; and mirrors and mirror members. This is done using the parameters in the `[Actions]` section, which is valid only in a merge file and does not appear in (and cannot be added to) an instance's CPF. For example, to add a global mapping to an instance, you would

include in the merge file an `[Actions]` section containing the `CreateMapGlobal` parameter. The parameters in this section, which create, modify, and delete system objects, are sometimes called *action parameters*, to distinguish them from those that update parameter values, which are known as *update parameters*.

Action parameters can be used to manage objects on both new and existing instances. The operations specified by action parameters are idempotent, meaning that they are executed only if they would result in a change. More specifically:

- A Create action is not executed if the specified object exists.
- A Modify action is not executed if the specified object does not exist. (If the object exists but the action does not add to/modify the properties of the object, the action is executed but to no effect.)
- A Delete action is not executed if the specified object does not exist.
- A Config action is not executed if the object exists *and* the action does not add to/modify the properties of that object; if the object does not exist, *or* it exists and the action adds to/modifies its properties, the action is executed.

Action parameters are very useful in deployment, for example to configure several deployed instances as a mirror using the `ConfigMirror` action and the `MirrorSetName` and `MirrorDBName` properties of the `CreateDatabase` action, enabling the new mirror to be fully operational, with mirrored databases in place, immediately following deployment. On the other hand, when used in reconfiguring an existing instance with the **iris merge** command, action parameters can enable you to immediately make changes that would otherwise require a Management Portal procedure or a class method call; notable examples are adding an arbiter to an existing mirror with adding a new database to an existing mirror using the `MirrorSetName` and `MirrorDBName` properties of the `ModifyDatabase` action, , which must be done on the running primary instance.

The operations performed by action parameters are effected by calling methods of classes in the Config and Security classes and in the SYS.Database and %SYSTEM.SQL.Security classes, as well as two SQL commands.

For examples of the use of action parameters, see Useful Parameters in Automated Deployment.

For lists of action parameters by object managed, by corresponding class, and in order of processing, as well as by name, see [Actions] Parameter Reference.

5 How can I use configuration merge?

There are two primary uses for configuration merge:

- Configuring multi-instance topologies and stand-alone instances during deployment
- Reconfiguring deployed multi-instance topologies and stand-alone instances

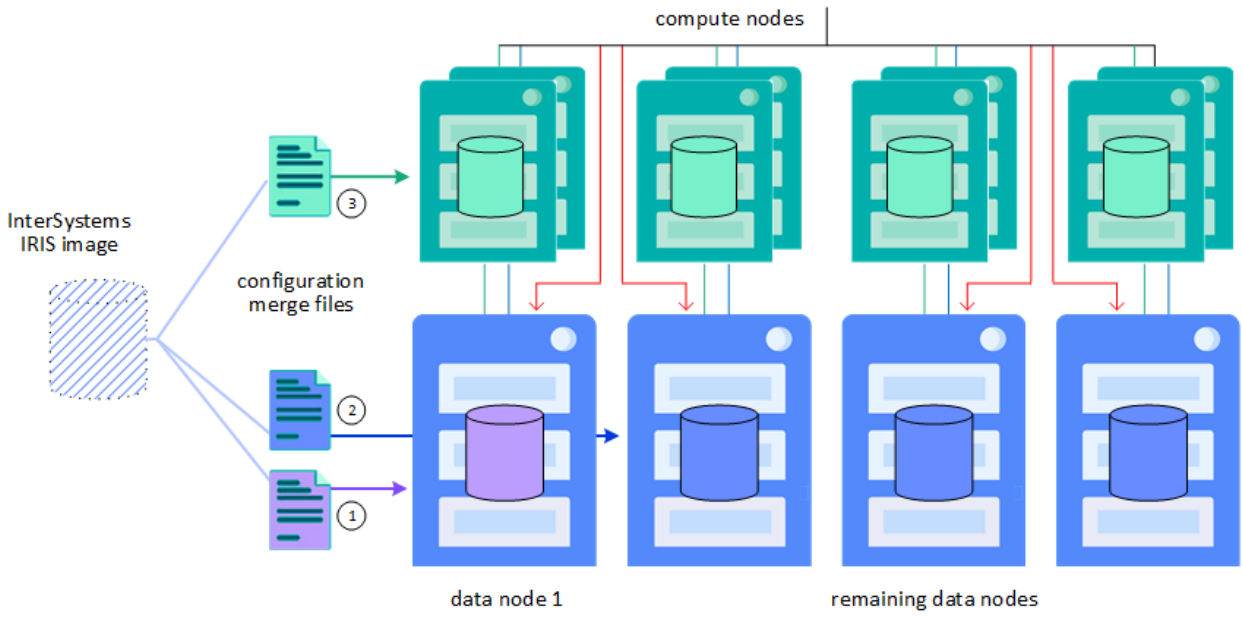
Regardless of the specific application of the configuration merge feature, InterSystems recommends keeping the merge files involved under version control to provide a record of all configuration changes, from deployment forward, over the life of an instance or a multi-instance topology.

When you incorporate configuration merge into your automated deployment or reconfiguration process, you can update the process by simply updating the merge files applied. Even in the case of individual instances used for purposes such as development and testing, users can be required get the latest version of the appropriate merge file before deploying or reconfiguring an instance, ensuring that its configuration matches a central specification. With version control, they can even return to an older configuration by selecting a previous version of the merge file.

6 Configuration merge in deployment

Applying a configuration merge file during deployment lets you modify the default configurations of the deployed instance before it starts for the first time. 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, as shown in the following illustration; when all of the instances are up and running, so is the sharded cluster.

Figure 2: Automated Deployment of a Sharded Cluster Using Configuration Merge



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.

Activating configuration merge during deployment requires only that the location of the merge file be specified by the environment variable **`ISC_CPF_MERGE_FILE`**, or by the field used for that purpose in the InterSystems Kubernetes Operator (IKO). For example, in manual or scripted deployment:

```
ISC_CPF_MERGE_FILE=/home/user/mergefiles/cmf_090821.cpf
```

The specific manner in which you specify the merge file depends on the deployment mechanism you are using and whether the instance is containerized or noncontainerized.

- Deploying an InterSystems IRIS container
- Installing InterSystems IRIS from a kit
- Deploying with the InterSystems Kubernetes Operator

6.1 Deploying an InterSystems IRIS container with a merge file

When deploying an InterSystems IRIS container, the environment variable and merge file can be included in the following ways:

- Include them in the script or docker-compose.yml file you are using for deployment.

  In the following sample deployment script, the merge file specified by **ISC_CPF_MERGE_FILE**, as well as the license key, are staged on the external volume specified for durable %SYS by **ISC_DATA_DIRECTORY** so they are accessible inside the container.

```
#!/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
  -p 9091:1972
  -v /data/durable263:/durable
  -h iris
  --name iris
  --cap-add IPC_LOCK
  --env ISC_DATA_DIRECTORY=/durable/irisdata
  --env ISC_CPF_MERGE_FILE=/durable/merge/CMF.cpf
  $container_image
  --key /durable/key/iris.key
```

  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.

  This sample docker-compose.yaml file contains the same elements as the deployment script.

```
version: '3.2'

services:
  iris:
    image: intersystems/iris:latest-em
    command: --key /durable/key/iris.key
    hostname: iris

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

    volumes:
    - /data/durable263:/durable

    environment:
    - ISC_DATA_DIRECTORY=/durable/irisdata
    - ISC_CPF_MERGE_FILE=/durable/merge/CMF.cpf
```

- Include the merge file in the container image and the environment variable in the script or Docker compose file used in deployment.

  When creating a custom InterSystems IRIS container image by starting with an InterSystems IRIS image from InterSystems and adding your own code and dependencies, you can execute the **iris merge** command in the Dockerfile to reconfigure the InterSystems IRIS instance contained in the image. For example, you can run the **iris merge** command with a merge file with `[Actions]` parameters to add namespaces and databases to the instance, which will then be present on the instance in every container created from your custom image. This method is explained and illustrated in Creating InterSystems IRIS Images in *Running InterSystems Products in Containers.*

  If you want the containerized instances to continue to use the merge file you placed in the container, you can set **ISC_CPF_MERGE_FILE** in your script or compose file to the location of this file. You can also run an additional, separate merge during deployment using a merge file positioned on the durable SYS volume, as illustrated above; if you plan to do this, add a command in the Dockerfile to remove the included merge file after it has been applied to the instance.

For examples of use cases for automated deployment using configuration merge, see Useful parameters in automated deployment.

6.2 Installing InterSystems IRIS from a kit with a merge file

To apply a merge file when installing InterSystems IRIS from a kit, manually or in a script, you must separate installation from startup, using the steps below.

On UNIX® or Linux system:

1. Install the instance without starting it by preceding the **irisinstall** or **irisinstall_silent** script with the **ISC_PACKAGE_STARTIRIS** parameter, as follows:

   ```
   ISC_PACKAGE_INSTANCENAME="IRIS27" ISC_PACKAGE_STARTIRIS="N" /tmp/iriskit/irisinstall
   ```

2. Start the instance with the **iris start** command, preceding it with the **ISC_CPF_MERGE_FILE** variable, as follows:

   ```
   ISC_CPF_MERGE_FILE=/tmp/iriskit/CMF/merge.cpf iris start IRIS27
   ```

On a Windows system:

1. Install and start the instance, for example:

   ```
   IRIS-2023.2.0.227.0-win_x64.exe /instance IRIS27 INSTALLDIR=C:\InterSystems\IRIS27
   ```

2. When the instance is fully started, use the **iris merge** command (as described in Reconfigure an existing instance using configuration merge) to apply your merge file:

   ```
   iris merge IRIS27 C:\InterSystems\IRIS27\merge.cpf
   ```

6.3 Using a merge file when deploying with the InterSystems Kubernetes Operator

Kubernetes is an open-source orchestration engine for automating deployment, scaling, and management of containerized workloads and services. 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.

When deploying with the IKO, you use a Kubernetes ConfigMap to integrate one or more merge files into the deployment process. For detailed information, see Create configuration files and provide a config map for them in *Using the InterSystems Kubernetes Operator*.

7 Reconfigure an existing instance using configuration merge

By automating application of the same merge file to multiple running instances, you can simultaneously reconfigure all of those instances in the same way, applying the same set of configuration changes across your application or cluster. You can avoid updating settings that may have been customized on a per-instance basis and should not be modified simply by omitting these from the merge file, while including only those you know it is safe and desirable to change. A single automated program can of course apply different merge files to different groups of instances (such as different mirror member or cluster nodes types) as described in the previous section.

Applying all configuration changes with a merge file helps you streamline the process of making changes and maintain greater control over the instance's configuration. Rather than making numerous individual changes from the Terminal, on multiple pages of the Management Portal, or by editing an instance's CPF manually, you can execute all the changes at once using identical syntax in a merge file. By keeping your merge files under version control, you ensure the availability of configuration history and the option of restoring a previous configuration.

The **iris merge** command, which can be used on both UNIX/Linux and Windows systems, applies a merge file to a running instance. For Windows operating systems, it requires OS authentication to execute. It is executed as follows:

```
iris merge instance [merge-file] [target-CPF]
```

where:

- *instance* is the name of the InterSystems IRIS instance.
- *merge-file* is the absolute or relative path to the merge file, including the filename. If *merge-file* is not specified, the value of the **ISC_CPF_MERGE_FILE** environment variable is used, if it is set.
- *target-CPF* is the absolute or relative path to the active CPF for instance *instance*, which is assumed to be named iris.cpf. If *target-CPF* is not specified, the defaults are as follows:
  - For noncontainerized instances, the iris.cpf file located in the directory specified by the **ISC_PACKAGE_INSTALLDIR** environment variable, if it is set. For most existing instances, this variable is not set, and you must explicitly specify the location of the target CPF.
  - For containerized instances, the iris.cpf file located in the directory specified by the **ISC_DATA_DIRECTORY** environment variable or, if it is not set (because durable %SYS is not in use), the **ISC_PACKAGE_INSTALLDIR** environment variable, which is always set in an InterSystems IRIS container.

No merge is performed if:

- The specified merge file is not present, or the *merge-file* argument is omitted and **ISC_CPF_MERGE_FILE** does not exist.
- There is no CPF in the specified target location, or if the target location is not specified and neither **ISC_DATA_DIRECTORY** or **ISC_PACKAGE_INSTALLDIR** exists.

An exit code of 3 indicates a successful merge.

Some changes merged into a CPF will not take effect immediately, but require a restart. For example, a change in the value of the gmheap parameter, which determines the size of the instance's shared memory heap, does not take effect until the instance is restarted. When your merge file contains one or more such parameters, you may need to apply the merge file as part of a restart, as in the following sample script excerpt:

```
# restart instance with  the necessary parameters (all on one line)
sudo ISC_CPF_MERGE_FILE=/net/merge_files/config_merge.cpf iris stop IRIS restart
```

On the other hand, applying a merge file with the **iris merge** command lets you immediately change settings that do not require a restart, including those that *cannot* be set during instance startup; an example, as noted in Can configuration merge customize more than the configuration?, is adding a database to an existing mirror.

Important: When a container is deployed with configuration merge (as described in Deploying an InterSystems IRIS container with a merge file), the merge file specified by **ISC_CPF_MERGE_FILE** (which is persistent in the container) is continuously monitored for updates as long as the container is running, with updates immediately merged by an **iris merge** command when they occur. This means that you can update the configuration of a containerized instance at any time by updating its merge file, making it easier to automate reconfiguration of containerized instances and clusters.

8 Managing configuration changes

In addition to the use of configuration merge in deployment or with an existing instance through the **iris merge** command or during a restart, an instance's CPF can be altered using the Management Portal, the Config.* classes, or a text editor. These methods are generally used for modifying individual settings on individual instances as needed, rather than reconfiguring multiple instances. If you use configuration merge to automatically deploy and/or reconfigure multiple instances, the strongly recommended best practice is to *confine all configuration changes to this method* — even when this means, for example, using **iris merge** merge to change just one or two parameters on one instance. That way, assuming you version and store the merge files you employ, you can maintain a record of the configuration of each instance through time and avoid the possibility of configuration merge overwriting changes made by other means.

In a container, the potential for the latter is very great due to the continuous monitoring and merging of the merge file identified by the **ISC_CPF_MERGE_FILE** variable, as described in the previous section. This allows you to use configuration merge and a central repository of merge files to apply further changes to existing instances simply by updating their merge files at any time. However, if the configuration parameters included in the merge file have been changed on the instance in the container by another method since deployment, the update merge can erase those changes, of which there may not be any record. Confining all configuration changes to configuration merge avoids this. (If the merge file does not exist, startup displays an error message and continues.)

If you do not confine changes to configuration merge, you can avoid the possibility of configuration merge making unwanted changes by including in your automation (using, for example, the **iris-main --after** option) the scripting of either or both of the following after instance startup:

- The deletion of the **ISC_CPF_MERGE_FILE** environment variable in each deployed container. (If the merge file does not exist, startup displays an error message and continues.)
- The replacement of the merge file in each container with an empty file.

9 Useful parameters in automated deployment

The configuration merge feature can be used to update any combination of settings in an instance's CPF and execute certain operations on the instance as specified in the `[Actions]` section. Several automated deployment use cases that you may find useful and make good examples of the power of the configuration merge feature, along with the parameters involved, are discussed in this section, including:

Update parameters

- Change the default password
- Configure and allocate memory
- Configure SQL and SQL Shell options and map SQL datatypes
- Update parameters example

Action parameters

- Create, modify and delete security objects
- Create, modify, and delete database objects
- Deploy a distributed cache cluster
- Mirror the cluster's data server

9.1 Update Parameters

The parameters described in the following sections are among those used to modify values in the deployed instance's CPF before the instance is started, thereby updating the default CPF in the deployment source (installation kit or container). Each parameter name provided is linked to its listing in the *Configuration Parameter File Reference* so you can easily review a parameter's purpose and details of its usage.

9.1.1 Change the Default Password

As described in Authentication and Passwords in *Running InterSystems Products in Containers*, you can use the PasswordHash setting in the `[Startup]` section to customize the default password of the predefined accounts on an instance at deployment, which eliminates the serious security risk entailed in allowing the default password of **SYS** to remain in effect. (The password of each predefined account should be individually changed following deployment.)

Table 1: Password Parameter

[Startup] Parameter	Specifies
PasswordHash	Default password for the predefined user accounts based on a cryptographic hash of the value and its salt

Note: The `[Actions]/CreateUser` parameter also takes a PasswordHash argument that is equivalent to the parameter (see [Actions] Parameter Reference).

9.1.2 Configure and Allocate Memory

There are a number of parameters affecting an InterSystems IRIS instance's memory usage, the optimal value of which can depend on the physical memory available, the instance's role within the cluster, the workload involved, and performance requirements.

For example, the optimal size of an instance's database cache, which can be specified using the globals parameter, can vary greatly depend on the instance's role; as noted above, sharded cluster data nodes typically require a relatively large cache. But even within that role, the optimal size depends on the size of the cluster's sharded data set, and the implemented size may be smaller than optimal due to resource constraints. (For more information, see Planning an InterSystems IRIS Sharded Cluster in the *Scalability Guide*.) Further, because the database cache should be carefully sized in general, the default database cache setting (the value of globals in the iris.cpf file provided in the container) is intentionally unsuitable for any production environment, regardless of the instance's role.

Some of the memory usage settings in the `[Config]` section of the CPF that you might want to update as part of deployment are listed in the following table:

Table 2: Memory Parameters

[Config] Parameter	Specifies
bbsiz	Maximum process private memory per process
globals	Shared memory allocated to the database cache (not from shared memory heap)
routines	Shared memory allocated to the routine cache (not from shared memory heap)
gmheap	Shared memory configured as the shared memory heap
jrnbufs	Shared memory allocated to journal buffers from the shared memory heap
locksiz	Maximum shared memory allocated to locks from the shared memory heap

For more detail on these and other memory-related parameters, see System Resource Planning and Management, Memory and Startup Settings, Configuring Journal Settings, and Monitoring Locks.

9.1.3 Configure SQL and SQL Shell Options and Map SQL Datatypes

You can specify the SQL and SQL Shell settings for instances you are deploying by merging one or more of the parameters in the `[SQL]` section of the CPF. In the Management Portal these settings can be reviewed and modified on the **SQL** page (**System Administration** > **Configuration** > **SQL and Object Settings** > **SQL**). You can map SQL system data types and SQL user data types to their InterSystems SQL equivalents on deployed instances using the `[SqlSysDatatypes]` and `[SqlUserDatatypes]` sections of the CPF, respectively. For more detail on SQL Shell setting and datatype mapping, see Configuring the SQL Shell and Data Types (SQL), respectively.

9.1.4 Update Parameter Example

The following sample CPF merge file includes some of the update parameters discussed in the preceding sections. The `SystemMode` parameter specifies a label that is displayed at the top of the Management Portal.

```
[Startup]
SystemMode=TEST
PasswordHash=FBFE8593AEFA510C27FD184738D6E865A441DE98,u4ocm4qh

[config]
bbsiz=-1
globals=0,0,900,0,0,0
routines=64
gmheap=256000
jrnbufs=96
locksiz=1179648

[SQL]
DefaultSchema=user
TimePrecision=6

[SqlSysDatatypes]
TIMESTAMP=%Library.PosixTime
```

9.2 Action Parameters

The parameters described in the following sections are among those that can be included in the `[Actions]` section to create, modify, or delete different types of objects on an instance as part deployment (or reconfiguration), including databases, namespaces, and mappings; users, roles, and resources; and many more. The use of action parameters (often simply called *actions*) is described in Can configuration merge customize more than the configuration?, which includes additional configuration merge file examples, and they are comprehensively listed in [Actions] Parameter Reference.

For comprehensive lists of all of the `[Actions]` parameters, see [Actions] Parameter Reference.

9.3 Create, Modify and Delete Security Objects

Include the operations described in the following table in the `[Actions]` section to create and modify security objects as part of deployment or reconfiguration

Table 3: Sample Security Object Creation Parameters

[Actions] Parameter	Specifies
`CreateUser`	The name and properties of the user account to be created. You can also use `ModifyUser` and `DeleteUser`.
`CreateRole`	The name and properties of the role to be created. You can also use `ModifyUser` and `DeleteUser`.
`CreateResource`	The name and properties of the resource to be created. You can also use `ModifyUser` and `DeleteUser`.
`GrantAdminPrivilege`, `GrantPrivilege`	The user account to grant SQL privileges and SQL admin privileges to, the privileges to be granted, and the namespace to grant them in. You can also use `RevokeAdminPrivlege` and `RevokePrivilege`.
`ModifyService`	The service to enable or disable.
`CreateApplication`	The name and properties of the application to be created. You can also use `ModifyApplication` and `DeleteApplication`.
`CreateSSLConfigs`	The name, location, and properties of the TLS/SSL configuration to be created. You can also use `ModifySSLConfigs` and `DeleteSSLConfigs`.
`CreateLDAPConfig`	The name and properties of the LDAP configuration to be created. You can also use `ModifyLDAPConfig` and `DeleteLDAPConfig`.
`CreateEvent`	The name, properties and status of the system audit event to be created. You can also use `ModifyEvent` and `DeleteEvent`.

To illustrate the use of action parameters with security objects, suppose you wanted to add to deployed instances a predefined account for a SQL administrator user who:

- Has SQL access through the %Service_Bindings service (%SQL role).
- Can read from or write to the **USER** database (%DB_USER role).
- Can create and drop tables, views, procedures, functions, methods, queries, and triggers (%DB_OBJECT_DEFINITION privilege) and use the BUILD INDEX command (%BUILD_INDEX privilege) in the **USER** namespace.

To do this, you could use the `CreateUser` parameter to create the user account with password and assign it the needed roles, and the `GrantAdminPrivilege` parameter to grant it the needed SQL privileges, as follows:

```
[Actions]

CreateUser:Name=SQLAdmin,
  PasswordHash="cec6638a357e7586fddfb15c0e7dd5719a1964e774cd37466fb0c49c05,
  323cb89148c887166dd2be61c107710539af2c01b43f07dccc8d030ac2c1a8cf7c5ace4a00d57e3780f,10000,SHA512",
  Roles="%SQL,%DB_USER"

GrantAdminPrivilege:Grantee=SQLAdmin,Namespace=USER,AdminPriv="%DB_OBJECT_DEFINITION,%BUILD_INDEX"
```

For information about the operations performed by these action parameters and the values of their properties, see Authentication and Passwords, About InterSystems Authorization, and SQL Users, Roles, and Privileges.

9.4 Create, Modify, and Delete Database Objects

Include the operations described in the following table in the `[Actions]` section to create databases (both local and remote), namespaces, and mappings as part of deployment or reconfiguration.

Table 4: Sample Database and Namespace Action Parameters

[Actions] Parameter	Specifies
`CreateDatabase`	The database's name and properties to be registered in InterSystems IRIS and the location on the host file system of the database file to be created. You can also use `ModifyDatabase` and `DeleteDatabase`.
`CreateDatabaseFile`	The location on the host file system of the database file to be created (without registering the database in InterSystems IRIS). You can also use `ModifyDatabaseFile` and `DeleteDatabaseFile`.
`CreateNamespace`	The name and properties of the namespace to be created in InterSystems IRIS. You can also use `ModifyNamespace` and `DeleteNamespace`.
`ModifyNamespace`	The name of the existing InterSystems IRIS namespace and its properties to be modified. You can also use `CreateNamespace` and `DeleteNamespace`.
`CreateMapGlobal`	The namespace to create the mapping in, the specification of the global to be mapped, and the database in which that global resides. You can also use `ModifyMapGlobal` and `DeleteMapGlobal`. In addition, you can use `Create`/`Modify`/`DeleteMapRoutine` and `Create`/`Modify`/`DeleteMapPackage` to create, modify, and delete routine and package mappings.

To illustrate the use of action parameters with database-related objects, suppose you wanted to:

- Create a database and a resource for it, then modify an existing namespace to make the database you created its global database and enable interoperability.
- Create a second database with resources, then create an interoperability-enabled namespace with the database as its default global database.

The following example shows how you could do this with a merge file:

```
[Actions]

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/database-path/APPA
ModifyNamespace:Name=APPA,Globals=APPA,Interop=1

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/database-path/APPB
CreateNamespace:Name=APPB,Globals=APPB,Interop=1
```

Suppose that at a later point you wanted to add mappings of globals and routines in database **APPA** to namespace **APPB**. You could do this with a merge file like the following:

```
[Actions]
CreateMapGlobal:Namespace=APPB,Name=global-spec,Database=APPA
CreateMapRoutine:Namespace=APPB,Name=routine-spec,Database=APPA
```

For information about the operations performed by these action parameters and the values of their properties, see Create/Modify a Namespace, Create a Local Database, and Add Global, Routine, and Package Mapping to a Namespace.

9.5 Deploy a Distributed Cache Cluster (Nonmirrored)

With a few simple changes to the merge file example in the previous section, you can create merge files to deploy a distributed cache cluster.

Deploy the Data Server

The merge file below would be used to deploy the nonmirrored data server. In addition to creating the application databases and their associated resources and namespaces, it does the following:

- Enables the ECP service with an action parameter.
- Uses an update parameter to set the maximum number of concurrent application server connections the data server can accept to 16.

Differences from the previous sample merge file are emphasized.

```
# nonmirrored data server merge file

[Config]

MaxServerConn=16

[Actions]

ModifyService:Name=%service_ecp,Enabled=1

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/database-path/APPA
CreateNamespace:Name=APPA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/database-path/APPB
CreateNamespace:Name=APPB,Globals=APPB
```

Deploy the Application Servers

This merge file, which would be used to deploy all of the application servers, does the following:

- Adds the data server as a remote server with an update parameter
- Modifies the `CreateDatabase` actions above by adding the `Server` and `LogicalOnly` properties and updating the `Directory` property to point to existing databases on the remote server, rather than a local directory in which to create a local database.

Differences from the sample merge file in the previous section are emphasized.

```
# app servers merge file

[ECPServers]

dataAB=dataserver-address,port,0

[Actions]

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Server=dataAB,Directory=/database-path-on-dataserver-dataAB/APPA,ResourceName=%DB_%APPA,
  LogicalOnly=1,
CreateNamespace:Name=APPDBA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Server=dataAB,Directory=/database-path-on-dataserver-dataAB/APPB,ResourceName=%DB_%APPB,
 LogicalOnly=1
CreateNamespace:Name=APPB,Globals=APPB
```

For information about the operations performed by these action parameters and the values of their properties, see Remote Databases and Deploying a Distributed Cache Cluster.

9.6 Mirror the Cluster's Data Server

To deploy one or more InterSystems IRIS mirrors, you can use separate configuration merge files, each containing the `ConfigMirror` action parameter, for the different mirror roles, sequentially deploying the first failover member(s), then the second failover member(s), then DR async members. (Reporting async members can be added to the mirror after deployment.)

You can also deploy using a single merge file and hostname matching, which determines which member to deploy on each of a set of hosts the names of which match the required pattern.

This section provides examples of each approach and a table listing the commonly-used properties of the `ConfigMirror` parameter.

For detailed information about mirror configuration, see Mirroring Architecture and Planning and Creating a Mirror. Be sure to read Mirroring with InterSystems IRIS Containers before planning containerized deployment of mirrors, or reconfiguring existing containerized instances into mirrors. Among other important considerations, you must ensure that the ISCAgent starts whenever the container for a failover or DR async mirror member starts.

9.6.1 Deploy the Mirror Using Separate Merge Files

In planning deployment using separate merge files it is important to bear in mind that the instance configured as the mirror primary must be running before other members can be added, so you must ensure that this instance is deployed and successfully started before other instances are deployed as the remaining members.

Deploy the Data Server Mirror Members

The following merge files deploy the distributed cache cluster's data server as a mirror with a DR async member by doing the following:

- Including the `ConfigMirror` action parameter to create the mirror and add members.
- On the primary, adding the created databases to the mirror (they will be automatically added on the other members).

Differences from the corresponding merge file in the previous section are emphasized.

```
# mirrored data server primary merge file

[Config]
MaxServerConn=16

[Actions]

ModifyService:Name=%service_ecp,Enabled=1

ConfigMirror:Name=CLUSTERAB,SSLDir=ssl-directory-path,
  Member=primary,Primary=localhost,ArbiterURL=address:port

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/ddatabase-path/APPA,
   MirrorSetName=CLUSTERAB,MirrorDBName=APPA
CreateNamespace:Name=APPA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/ddatabase-path/APPB,
   MirrorSetName=CLUSTERAB,MirrorDBName=APPB
CreateNamespace:Name=APPB,Globals=APPB

# mirrored data server backup/DR async merge file;
# for ConfigMirror Member property enter either =backup or =drasync as appropriate

[Config]
MaxServerConn=16

[Actions]

ModifyService:Name=%service_ecp,Enabled=1

ConfigMirror:Name=CLUSTERAB,SSLDir=ssl-directory-path,
  Member=backup|drasync,Primary=primary-address:,ArbiterURL=address:port

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/ddatabase-path/APPA,
   MirrorSetName=CLUSTERAB,MirrorDBName=APPA
CreateNamespace:Name=APPA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/ddatabase-path/APPB,
   MirrorSetName=CLUSTERAB,MirrorDBName=APPB
CreateNamespace:Name=APPB,Globals=APPB
```

Note: To copy existing databases to the mirror created by the `ConfigMirror` action instead of creating empty ones, you could use the `Seed` parameter of the `CreateDatabase` action to specify the pathnames of the databases to copy, as shown in this modified excerpt from either section (primary or backup/DR async) of the previous example:

```
ConfigMirror:Name=CLUSTERAB,SSLDir=ssl-directory-path,
  Member=backup|drasync,Primary=primary-address:,ArbiterURL=address:port

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/ddatabase-path/APPA,
  MirrorSetName=CLUSTERAB,MirrorDBName=APPA,Seed=/mnt/databases/DB1
CreateNamespace:Name=APPA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/ddatabase-path/APPB,
  MirrorSetName=CLUSTERAB,MirrorDBName=APPB,Seed=/mnt/databases/DB2
CreateNamespace:Name=APPB,Globals=APPB
```

Deploy the Application Servers with Mirrored Data Server

This merge file alters the application server merge file previously shown only by changing the `0` at the end of the remote server definition action in `[ECPServers]` to `1`, as emphasized, to indicate that the remote server is a mirror, which allows application connections to transparently switch to the new primary after failover.

```
# app servers merge file

[ECPServers]

dataAB=dataserver-address,port,1

[Actions]

CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/database-path-on-dataserver-dataAB/APPA,ResourceName=%DB_%APPA,
  Server=dataAB,LogicalOnly=1
CreateNamespace:Name=APPDBA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/database-path-on-dataserver-dataAB/APPB/,ResourceName=%DB_%APP,B
  Server=dataAB,LogicalOnly=1
CreateNamespace:Name=APPB,Globals=APPB
```

9.6.2 Deploy the Mirror Using Hostname Matching

You can automatically deploy one or more mirrors from a single merge file if the deployment hosts have names ending in *-number* (or, as a regular expression, .*-[0-9]+$), for example iris-000, iris-001, iris-002 ..., or in *-number-number*, for example iris-0-0, iris-0-1, iris-1-0, iris-1-1 .... You do this by

- Setting the `Map` property (not used with the separate merge file approach) to the pattern you want (it is `primary,backup` by default, but can also contain up to 14 DR async members as in `primary,backup,drasync,...`)
- Setting the `Member` and `Primary` properties to `auto`.

For example, if you used the following `ConfigMirror` action parameter, mirrors members would be deployed on appropriately named hosts as shown in the table after the example:

```
ConfigMirror:Name=AUTOMIRROR,SSLDir=ssl-directory-path,
  Map="primary,backup,drasync",
  Member=auto,Primary=auto,ArbiterURL=address:port
```

Table 5: Mirror Deployment by Hostname

Single-number hotnames	Double-number hostnames	Mirror member role
mirror-000	**mirror-0-0**	primary
mirror-002	**mirror-0-1**	backup
mirror-003	**mirror-0–2**	DR async
mirror-005	**mirror-1-0**	primary
mirror-006	**mirror-1-1**	backup
mirror-007	**mirror-1-2**	DR async

Deploy Mirror Members by Hostname

Incorporating a `ConfigMirror` action like the one above, you could use hostname matching to deploy a three-member mirrored data server using the following single merge file on three sequentially named hosts such as iris-001, iris-002, and iris-003 rather than three merge files as shown in the previous section.

```
# mirrored data server using single merge file and hostname mapping

[Config]
MaxServerConn=16

[Actions]

ModifyService:Name=%service_ecp,Enabled=1

ConfigMirror:Name=CLUSTERAB,SSLDir=ssl-directory-path,
  Map=”primary,backup,drasync”,Member=auto,
  Primary=auto,ArbiterURL=address:port
CreateResource:Name=%DB_%APPA,Description="APPA database"
CreateDatabase:Name=APPA,Directory=/ddatabase-path/APPA,
  MirrorSetName=CLUSTERAB,MirrorDBName=APPA
CreateNamespace:Name=APPA,Globals=APPA

CreateResource:Name=%DB_%APPB,Description="APPB database"
CreateDatabase:Name=APPB,Directory=/ddatabase-path/APPB,
  MirrorSetName=CLUSTERAB,MirrorDBName=APPB
CreateNamespace:Name=APPB,Globals=APPB
```

9.6.3 ConfigMirror Properties

The following table shows the most commonly used properties of the `ConfigMirror` action parameter. In previous releases of InterSystems IRIS, most of these were parameters in the `[Startup]` section of the CPF; accordingly, the name of the corresponding former `[Startup]` parameter is shown in the table.

Table 6: ConfigMirror Properties

Property	Deploying using separate merge files	Deploying using a single merge file and hostname mapping
`Name` (formerly `[Startup]/MirrorSetName`)	Name of the new mirror (when deploying a primary) or the mirror to join (when deploying a backup or DR async)	Name of mirror
`Map`	(not used)	Sets the pattern used to match mirror members with hostnames; default is `Map="primary,backup"`
`Member` (formerly `[Startup]/MirrorMember`)	Mirror member role at deployment (`primary`, `backup`, or `drasync`)	Set to `auto` to automatically match mirror members to hostnames
`Primary` (formerly `[Startup]/MirrorPrimary`)	Name or IP address of the primary's host	Set to `auto` to automatically match mirror members to hostnames
`SSLDir` (formerly `[Startup]/MirrorSSLDir`)	Location on the host of the mirror TLS/SSL configuration for the instance, a directory containing the required Certificate Authority certificate (CAFile.pem), local certificate (CertificateFile.pem), and private key file (PrivateKeyFile.pem).	(see previous column; TLS configurations must be identically located on all hosts)
`ArbiterURL` (formerly `[Startup]/ArbiterURL`)	Host (hostname or IP address) and port of the arbiter to be configured for the mirror (when deploying the primary) or configured for existing primary (when deploying a backup or DR async)	Host (hostname or IP address) and port of the arbiter configured for the mirror