Scalability Guide

Version 2024.2
2024-09-05

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

Scalability Guide
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

1

InterSystems IRIS Scalability Overview

Today's data platforms are called on to handle a wide variety of workloads. As a workload of any type grows, a data platform must be able to scale to meet its increasing demands, while at the same time maintaining the performance standards the enterprise relies on and avoiding business disruptions.

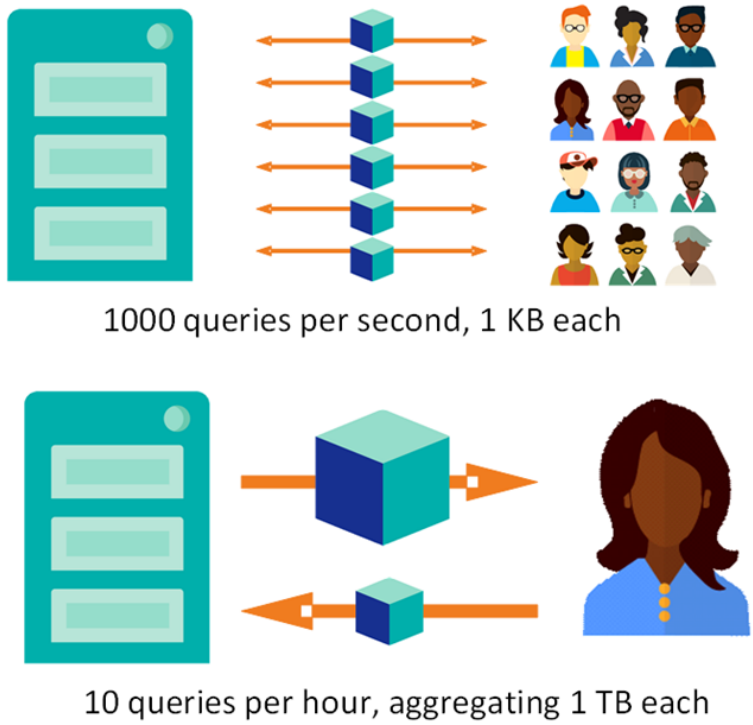
This page reviews the scaling features of InterSystems IRIS® data platform, and provides guidelines for a first-order evaluation of scaling approaches for your enterprise data platform.

1.1 Scaling Matters

What you do matters, and whether you care for ten million patients, process billions of financial orders a day, track a galaxy of stars, or monitor a thousand factory engines, your data platform must not only support your current operations but enable you to *scale* to meet increasing demands. Each business-specific workload presents a different challenge to the data platform on which it operates — and as a business grows, that challenge becomes even more acute.

For example, consider the two situations in the following illustration:

Figure 1–1: Comparing Workloads



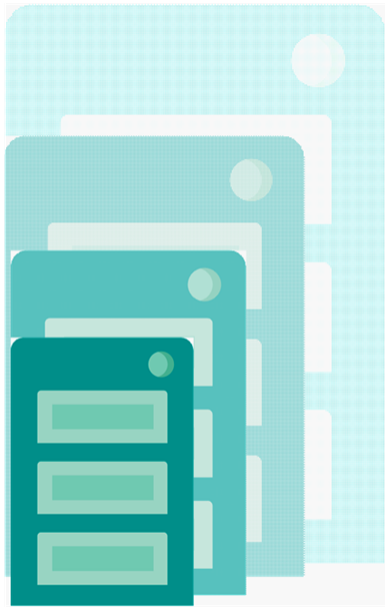
Both workloads are demanding, but it is hard to say which is more demanding — or how to scale to meet those demands.

We can better understand data platform workloads and what is required to scale them by decomposing them into components that can be independently scaled. One simplified way to break down these workloads is to separate the components of *user volume* and *data volume*. A workload can involve many users interacting frequently with a relatively small database, as in the first example, or fewer requests from what could be a single user or process but against massive datasets, like the second. By considering user volume and data as separate challenges, we can evaluate different scaling options. (This division is a simplification, but one that is useful and easy to work with. There are many examples of complex workloads to which it is not easily applied, such as one involving many small data writers and a few big data consumers.)

1.2 Vertical Scaling

The first and most straightforward way of addressing more demanding workloads is by scaling up — that is, taking advantage of vertical scalability. In essence, this means making an individual server more powerful so it can keep up with the workload.

Figure 1–2: Vertical Scaling



In detail, vertical scaling requires expansion of the capacity of an individual server by adding hardware components that alleviate the workload bottlenecks you are experiencing. For example, if your cache can't handle the working set required by your current user and data volume, you can add more memory to the machine.

Vertical scaling is generally well understood and architecturally straightforward; with good engineering support it can help you achieve a finely tuned system that meets the workload's requirements. It does have its limits, however:

- Today's servers with their hundred-plus CPU cores and memory in terabytes are very powerful, but no matter what its capacity, a system can simultaneously create and maintain only so many sockets for incoming connections.
- Premium hardware comes at a premium price, and once you've run out of sockets, replacing the whole system with a bigger, more expensive one may be your only option.
- Effective vertical scaling requires careful sizing before the fact. This may be straightforward in a relatively static business, but under dynamic circumstances with a rapidly growing workload it can be difficult to predict the future.
- Vertical scaling does not provide elasticity; having scaled up, you cannot scale down when changes in your workload would allow it, which means you are paying for excess capacity.
- Vertical scaling stresses your software, which must be able to cope effectively and efficiently with the additional hardware power. For example, scaling to 128 cores is of little use if your application can handle only 32 processes.

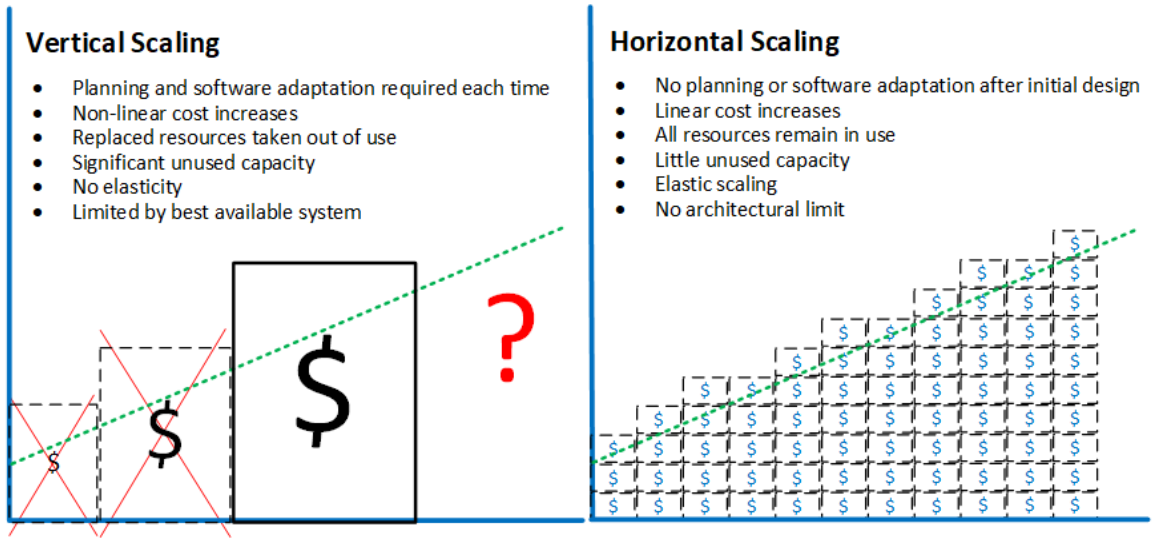
For more information on vertically scaling InterSystems IRIS data platform, see System Resource Planning and Management and General Performance Enhancement on InterSystems IRIS Platforms.

1.3 Horizontal Scaling

When vertical scaling does not provide the complete solution — for example, when you hit the inevitable hardware (or budget) ceiling — or as an alternative to vertical scaling, some data platform technologies can also be scaled horizontally by clustering a number of smaller servers. That way, instead of adding specific components to a single expensive server, you can add more modest servers to the cluster to support your workload as volume increases. Typically, this implies dividing the single-server workload into smaller pieces, so that each cluster node can handle a single piece.

Horizontal scaling is financially advantageous both because you can scale using a range of hardware, from dozens of inexpensive commodity systems to a few high-end servers to anywhere in between, and because you can do so gradually, expanding your cluster over time rather than the abrupt decommissioning and replacement required by vertical scaling. Horizontal scaling also fits very well with virtual and cloud infrastructure, in which additional nodes can be quickly and easily provisioned as the workload grows, and decommissioned if the load decreases.

Figure 1–3: Horizontal Scaling Addresses Vertical Scaling's Limitations



On the other hand, horizontal clusters require greater attention to the networking component to ensure that it provides sufficient bandwidth for the multiple systems involved. Horizontal scaling also requires significantly more advanced software, such as InterSystems IRIS, to fully support the effective distribution of your workload across the entire cluster. InterSystems IRIS accomplishes this by providing the ability to scale for both increasing user volume and increasing data volume.

1.3.1 Horizontal Scaling for User Volume

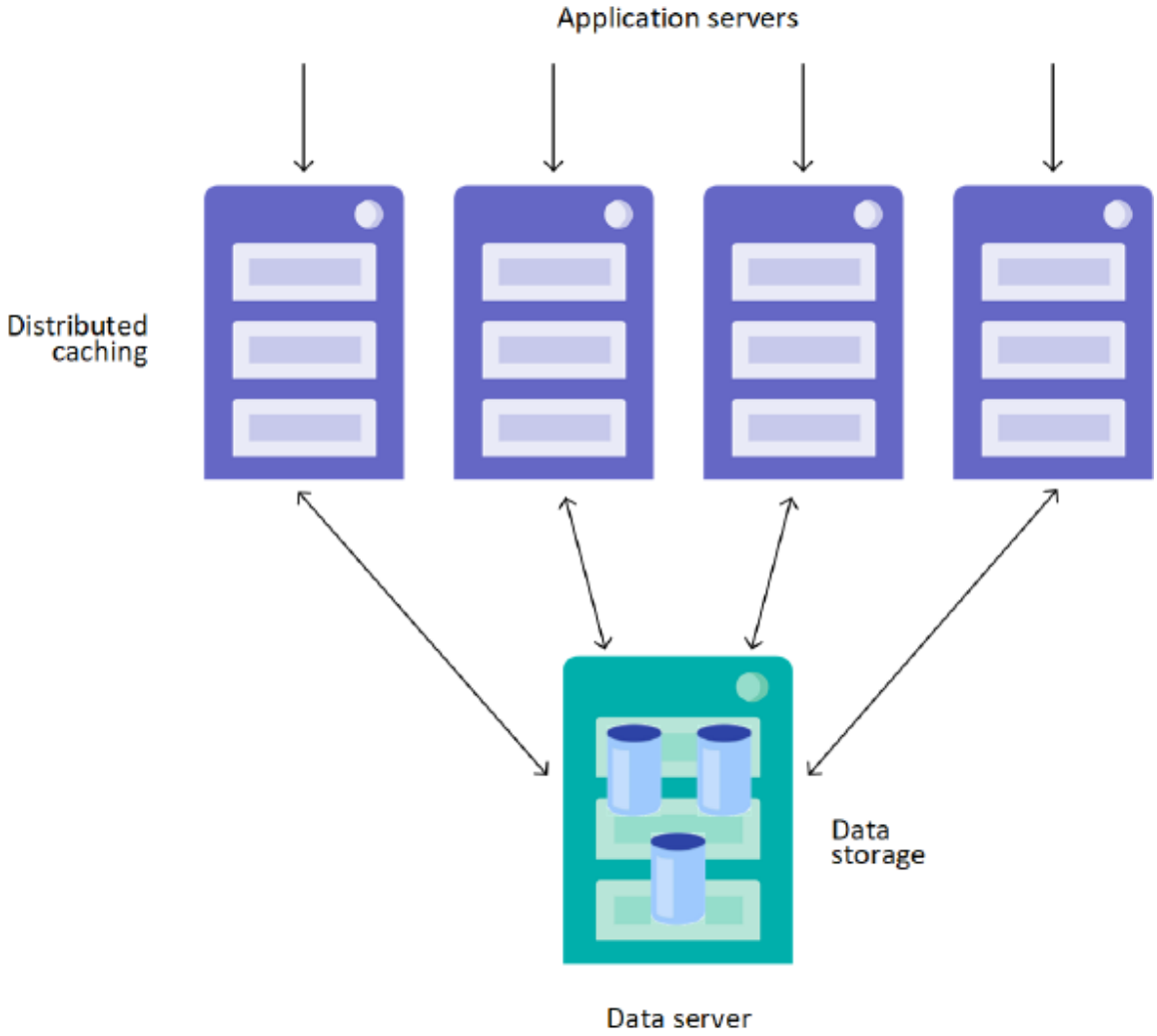
How can you scale horizontally when user volume is getting too big to handle with a single system at an acceptable cost? The short answer is to divide the user workload by connecting different users to different cluster nodes that handle their requests.

Figure 1–4: Dividing the User Workload

You can do this by using a load balancer to distribute users round-robin, but grouping users with similar requests (such as users of a particular application when multiple applications are in use) on the same node is more effective due to *distributed caching*, in which users can take advantage of each other's caches.

InterSystems IRIS provides an effective way to accomplish this through distributed caching, an architectural solution supported by the Enterprise Cache Protocol (ECP) that partitions users across a tier of application servers sitting in front of your data server. Each application server handles user queries and transactions using its own cache, while all data is stored on the data server, which automatically keeps the application server caches in sync. Because each application server maintains its own independent working set in its own cache, adding more servers allows you to handle more users.

Figure 1–5: InterSystems IRIS Distributed Cache Cluster



Distributed caching is entirely transparent to the user and the application code.

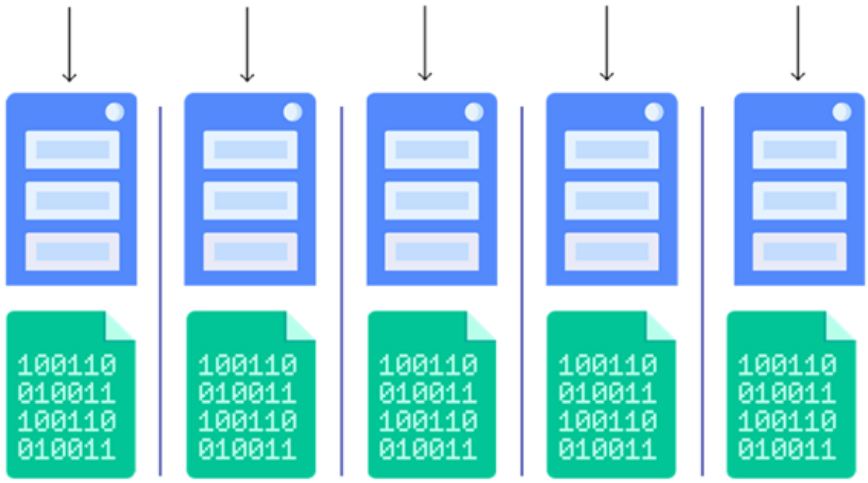
For more information on horizontally scaling InterSystems IRIS data platform for user volume, see Horizontally Scaling for User Volume with Distributed Caching.

1.3.2 Horizontal Scaling for Data Volume

The data volumes required to meet today's enterprise needs can be very large. More importantly, if they are queried repeatedly, the working set can get too big to fit into the server's cache; this means that only part of it can be kept in the cache and disk reads become much more frequent, seriously impacting query performance.

As with user volume, you can horizontally scale for data volume by dividing the workload among several servers. This is done by partitioning the data.

Figure 1–6: Partitioning the Data Workload



InterSystems IRIS achieves this through its sharding capability. An InterSystems IRIS *sharded cluster* partitions data storage, along with the corresponding caches, across a number of servers, providing horizontal scaling for queries and data ingestion while maximizing infrastructure value through highly efficient resource utilization.

In a basic sharded cluster, a sharded table is partitioned horizontally into roughly equal sets of rows called shards, which are distributed across a number of data nodes. For example, if a table with 100 million rows is partitioned across four data nodes, each stores a shard containing about 25 million rows. Nonsharded tables reside wholly on the first data node configured.

Queries against a sharded table are decomposed into multiple *shard-local queries* to be run in parallel on the data nodes; the results are then combined and returned to the user. This distributed data layout can further be exploited for parallel data loading and with third party frameworks.

In addition to parallel processing, sharding improves query performance by partitioning the cache. Each data node uses its own cache for shard-local queries against the data it stores, making the cluster's cache for sharded data roughly as large as the sum of the caches of all the data nodes. Adding a data node means adding dedicated cache for more data.

As with application server architecture, sharding is entirely transparent to the user and the application.

Sharding comes with some additional options that greatly widen the range of solutions available, including the following:

- Mirroring

  InterSystems IRIS mirroring can be used to provide high availability for data nodes.

- Compute nodes

  For advanced use cases in which low latencies are required, potentially at odds with a constant influx of data, *compute nodes* can be added to provide a transparent caching layer for servicing queries. Compute nodes support query execution only, caching the sharded data of the data nodes to which they are assigned (as well as nonsharded data when necessary). When a cluster includes compute nodes, read-only queries are automatically executed on them, while all write operations (insert, update, delete, and DDL operations) are executed on the data nodes. This separation of query workload and data ingestion improves the performance of both, and assigning multiple compute nodes per data node can further improve the query throughput and performance of the cluster.

For more information on horizontally scaling InterSystems IRIS data platform for data volume, see Horizontally Scaling for Data Volume with Sharding.

1.3.3 Automated Deployment of Horizontally Scaled Configurations

InterSystems IRIS provides several methods for automated cluster deployment, which can significantly simplify the deployment process, especially for complex horizontal cluster configurations. For more information about automated cluster deployment, see Automated Deployment Methods for Clusters.

1.4 Evaluating Your Workload for Scaling Solutions

The subsequent topics cover the individual scalability features of InterSystems IRIS in detail, and you should consult these before beginning the process of scaling your data platform. However, the tables below summarize the overview, and provide some general guidelines concerning the scaling approach that might be of the most benefit in your current circumstances.

Table 1–1: InterSystems IRIS Scaling Pros and Cons

Scaling Approach	Pros	Cons
Vertical	Architectural simplicity Hardware is finely tuned to workload	Price/performance ratio is nonlinear Persistent hardware limitations Careful initial sizing required One-way scaling only
Horizontal	Price/performance ratio is more linear Can leverage commodity, virtual and cloud-based systems Elastic scaling	Emphasis on networking

Table 1–2: InterSystems IRIS Vertical Scaling Solutions

Conditions	Possible Solutions
High multiuser query volume: insufficient computing power, throughput inadequate for query volume.	Add CPU cores. Take advantage of parallel query execution to leverage high core counts for queries spanning a large dataset.
High data volume: insufficient memory, database cache inadequate for working set.	Add memory and increase cache size to leverage larger memory. Take advantage of parallel query execution to leverage high core counts.
Other insufficient capacity: bottlenecks in other areas such as network bandwidth.	Increase other resources that may be causing bottlenecks.

Table 1–3: InterSystems IRIS Horizontal Scaling Solutions

Conditions	Possible Solutions
High multiuser query volume: frequent queries from large number of users.	Deploy application server configuration (distributed caching).
High data volume: some combination of: • High volume and/or high rate of data ingestion • Large data sets • Complex queries involving large amounts of data processing (see Evaluating the Benefits of Sharding)	Deploy sharded cluster (partitioned data and partitioned caching), possibly adding compute nodes to separate queries from data ingestion and increase query throughput (see Deploy Compute Nodes)

1.5 General Performance Enhancement on InterSystems IRIS Platforms

The following information may be helpful in improving the performance of your InterSystems IRIS deployment.

- Simultaneous multithreading

  In most situations, the use of Intel Hyper-Threading or AMD Simultaneous Multithreading (SMT) is recommended for improved performance, either within a physical server or at the hypervisor layer in virtualized environments. There may be situations in a virtualized environment in which disabling Hyper-Threading or SMT is warranted; however, those are exceptional cases specific to a given application.

  In the case of IBM AIX®, IBM Power processors offer multiple levels of SMT at 2, 4, and 8 threads per core. With the latest IBM Power9 and Power10 processors, SMT-8 is the level most commonly used with InterSystems IRIS. There may be cases, however, especially with previous generation Power7 and Power8 processors, in which SMT-2 or SMT-4 is more appropriate for a given application. Benchmarking the application is the best approach to determining the ideal SMT level for a specific deployment.

- Semaphore allocation

  By default, InterSystems IRIS allocates the minimum number of semaphore sets by maximizing the number of semaphores per set (see Semaphores in InterSystems Products). However, this is some evidence that this is not ideal for performance on Linux systems with non-uniform memory access (NUMA) architecture.

  To address this, the semsperset parameter in the configuration parameter file (CPF) can be used to specify a lower number of semaphores per set. By default, semsperset is set to 0, which specifies the default behavior. Determining the most favorable setting will likely require some experimentation; if you have InterSystems IRIS deployed on a Linux/NUMA system, InterSystems recommends that you try an initial value of 250.

1.6 See Also

- Horizontally Scaling for User Volume with Distributed Caching

- Horizontally Scaling for Data Volume with Sharding
- System Resource Planning and Management
- General Performance Enhancement on InterSystems IRIS Platforms

2

Horizontally Scaling for User Volume with Distributed Caching

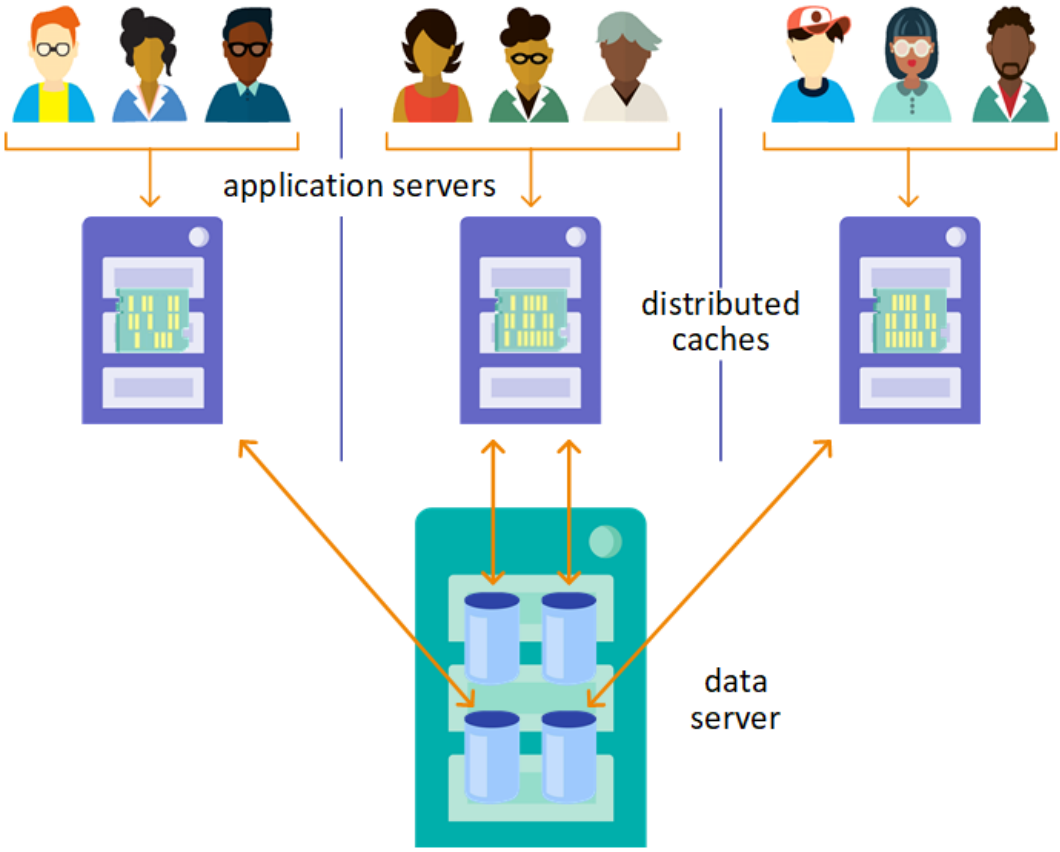
When vertical scaling alone proves insufficient for scaling your InterSystems IRIS data platform to meet your workload's requirements, you can consider *distributed caching*, an architecturally straightforward, application-transparent, low-cost approach to horizontal scaling.

2.1 Overview of Distributed Caching

The InterSystems IRIS distributed caching architecture scales horizontally for user volume by distributing both application logic and caching across a tier of application servers sitting in front of a data server, enabling partitioning of users across this tier. Each application server handles user requests and maintains its own database cache, which is automatically kept in sync with the data server, while the data server handles all data storage and management. Interrupted connections between application servers and data server are automatically recovered or reset, depending on the length of the outage.

Distributed caching allows each application server to maintain its own, independent working set of the data, which avoids the expensive necessity of having enough memory to contain the entire working set on a single server and lets you add inexpensive application servers to handle more users. Distributed caching can also help when an application is limited by available CPU capacity; again, capacity is increased by adding commodity application servers rather than obtaining an expensive processor for a single server.

Figure 2–1: Distributed Cache Cluster



This architecture is enabled by the use of the Enterprise Cache Protocol (ECP), a core component of InterSystems IRIS data platform, for communication between the application servers and the data server.

The distributed caching architecture and application server tier are entirely transparent to the user and to application code. You can easily convert an existing standalone InterSystems IRIS instance that is serving data into the data server of a cluster by adding application servers.

The following sections provide more details about distributed caching:

- Distributed Caching Architecture
- ECP Features
- ECP Recovery
- Distributed Caching and High Availability

2.1.1 Distributed Caching Architecture

To better understand distributed caching architecture, review the following information about how data is stored and accessed by InterSystems IRIS:

- InterSystems IRIS stores data in a file in the local operating system called a *database*. An InterSystems IRIS instance may (and usually does) have multiple databases.
- InterSystems IRIS applications access data by means of a *namespace*, which provides a logical view of the data stored in one or more databases. A InterSystems IRIS instance may (and usually does) have multiple namespaces.

- Each InterSystems IRIS instance maintains a *database cache* — a local shared memory buffer used to cache data retrieved from the databases, so that repeated instances of the same query can retrieve results from memory rather than storage, providing a very significant performance benefit.

The architecture of a distributed cache cluster is conceptually simple, using these elements in the following manner:

- An InterSystems IRIS instance becomes an application server by adding another instance as a *remote server*, and then adding any or all of its databases as *remote databases*. This makes the second instance a data server for the first instance.
- Local namespaces on the application server are mapped to remote databases on the data server in the same way they are mapped to local databases. The difference between local and remote databases is entirely transparent to an application querying a namespace on the application server.
- The application server maintains its own database cache in the same manner as it would if using only local databases. ECP efficiently shares data, locks, and executable code among multiple InterSystems IRIS instances, as well as synchronizing the application server caches with the data server.

In practice, a distributed cache cluster of multiple application servers and a data server works as follows:

- The data server continues to store, update, and serve the data. The data server also synchronizes and maintains the coherency of the application servers' caches to ensure that users do not receive or keep stale data, and manages locks across the cluster.
- Each query against the data is made in a namespace on one of the various application servers, each of which uses its own individual database cache to cache the results it receives; as a result, the total set of cached data is distributed across these individual caches. If there are multiple data servers, the application server automatically connects to the one storing the requested data. Each application server also monitors its data server connections and, if a connection is interrupted, attempts to recover it.
- User requests can be distributed round-robin across the application servers by a load balancer, but the most effective approach takes full advantage of distributed caching by directing users with similar requests to the same application server, increasing cache efficiency. For example, a health care application might group clinical users who run one set of queries on one application server and front-desk staff running a different set on another. If the cluster handles multiple applications, each application's users can be directed to a separate application server. The illustrations that follow compare a single InterSystems IRIS instance to a cluster in which user connections are distributed in this manner. (Load balancing user requests can even be detrimental in some circumstances; for more information see Evaluate the Effects of Load Balancing User Requests.)
- The number of application servers in a cluster can be increased (or reduced) without requiring other reconfiguration of the cluster or operational changes, so you can easily scale as user volume increases.

Figure 2–2: Local databases mapped to local namespaces on a single InterSystems IRIS instance

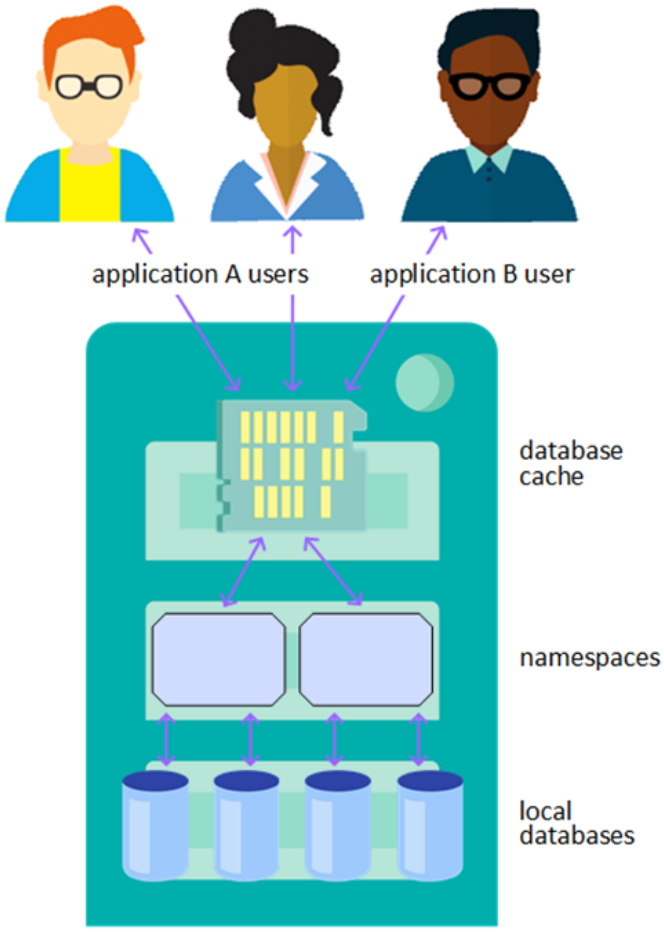
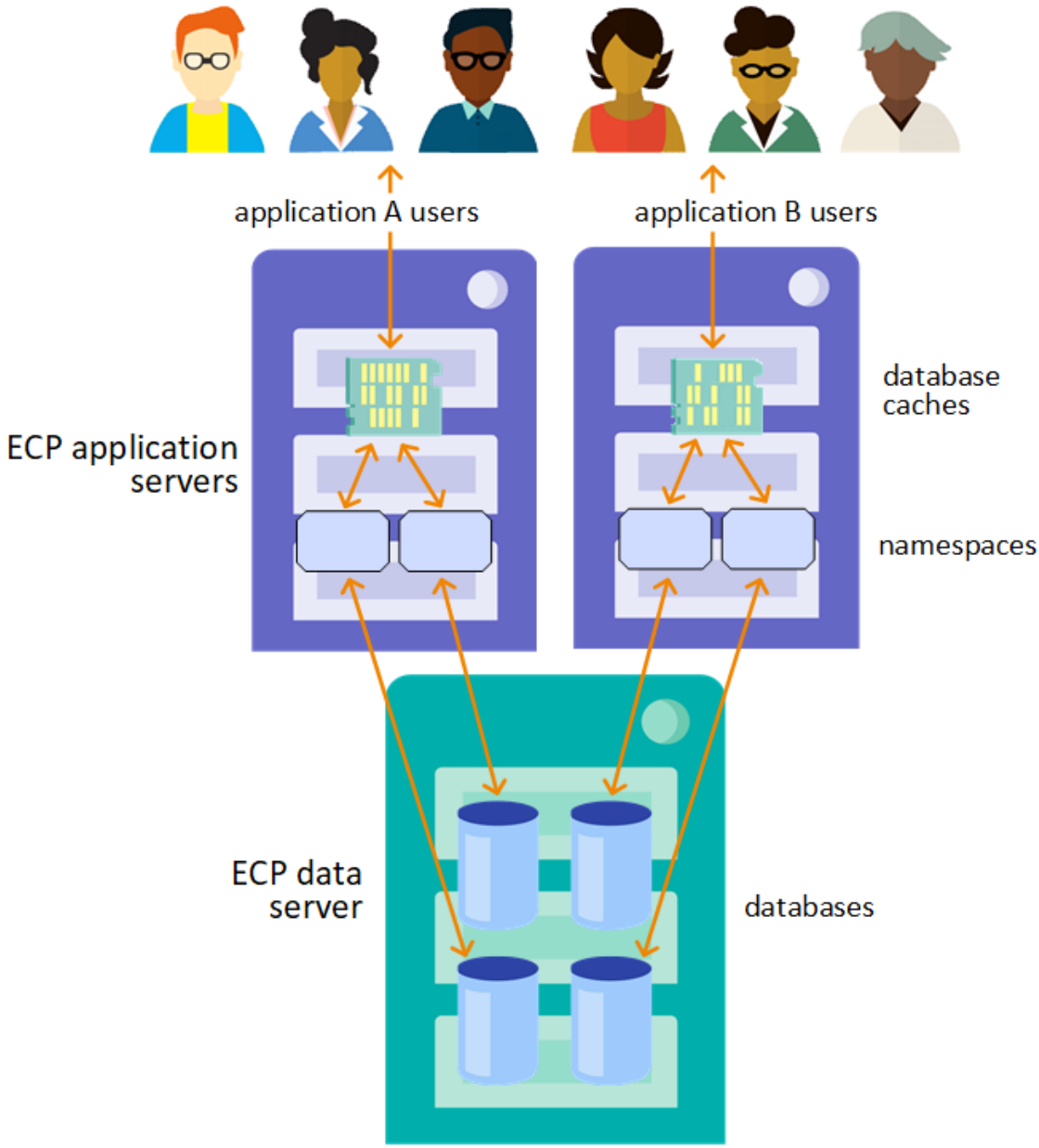


Figure 2–3: Remote databases on a data server mapped to namespaces on application servers in a distributed cache cluster



In a distributed cache cluster, the data server is responsible for the following:

- Storing data in its local databases.
- Synchronizing the application server database caches with the databases so the application servers do not see stale data.
- Managing the distribution of locks across the network.
- Monitoring the status of all application servers connections and taking action if a connection is interrupted for a specific amount of time (see ECP Recovery).

In a distributed cache cluster, each application server is responsible for the following:

- Establishing connections to a specific data server whenever an application requests data that is stored on that server.

- Maintaining, in its cache, data retrieved across the network.
- Monitoring the status of all connections to the data server and taking action if a connection is interrupted for a specific amount of time (see ECP Recovery).

Note: A distributed cache cluster can include more than one data server (although this is uncommon). An InterSystems IRIS instance can simultaneously act as both a data server and an application server, but cannot act as a data server for the data it receives as an application server.

2.1.2 ECP Features

ECP supports the distributed cache architecture by providing the following features:

- *Automatic, fail-safe operation.* Once configured, ECP automatically establishes and maintains connections between application servers and data servers and attempts to recover from any disconnections (planned or unplanned) between application server and data server instances (see ECP Recovery). ECP can also preserve the state of a running application across a failover of the data server (see Distributed Caching and High Availability).

  Along with keeping data available to applications, these features make a distributed cache cluster easier to manage; for example, it is possible to temporarily take a data server offline or fail over as part of planned maintenance without having to perform any operations on the application server instances.
- *Heterogeneous networking.* InterSystems IRIS systems in a distributed cache cluster can run on different hardware and operating system platforms. ECP automatically manages any required data format conversions.
- *A robust transport layer based on TCP/IP.* ECP uses the standard TCP/IP protocol for data transport, making it easy to configure and maintain.
- *Efficient use of network bandwidth.* ECP takes full advantage of high-performance networking infrastructures.

2.1.3 ECP Recovery

ECP is designed to automatically recover from interruptions in connectivity between an application server and the data server. In the event of such an interruption, ECP executes a recovery protocol that differs depending on the nature of the failure and on the configured timeout intervals. The result is that the connection is either recovered, allowing the application processes to continue as though nothing had happened, or reset, forcing transaction rollback and rebuilding of the application processes.

For more information on ECP connections, see Monitoring Distributed Applications; for more information on ECP recovery, see ECP Recovery Protocol and ECP Recovery Process, Guarantees, and Limitations.

2.1.4 Distributed Caching and High Availability

While ECP recovery handles interrupted application server connections to the data server, the application servers in a distributed cache cluster are also designed to preserve the state of the running application across a failover of the data server. Depending on the nature of the application activity and the failover mechanism, some users may experience a pause until failover completes, but can then continue operating without interrupting their workflow.

Data servers can be mirrored for high availability in the same way as a stand-alone InterSystems IRIS instance, and application servers can be set to automatically redirect connections to the backup in the event of failover. (It is not necessary or even possible to mirror an application server, as it does not store any data.) For detailed information about the use of mirroring in a distributed cache cluster, see Configuring ECP Connections to a Mirror.

The other failover strategies detailed in Failover Strategies for High Availability can also be used in a distributed cache cluster. Regardless of the failover strategy employed for the data server, the application servers reconnect and recover their states following a failover, allowing application processing to continue where it left off prior to the failure.

2.2 Deploying a Distributed Cache Cluster

An InterSystems IRIS distributed cache cluster consists of a data server instance providing data to one or more application servers instances, which in turn provide it to the application. InterSystems IRIS data platform offers several methods for automated deployment of distributed cache clusters. The instructions in this section are for manually deploying the cluster using the Management Portal. Information about securing the cluster after deployment is provided in Distributed Cache Cluster Security.

Important: In general, the InterSystems IRIS instances in a distributed cache cluster can be of different versions, as long as none of the application servers are of a later version than the data server. For important requirements and limitations regarding version compatibility, see ECP Interoperability.

While the data server and application server hosts can be of different operating systems and/or endianness, all InterSystems IRIS instances in a distributed cache cluster must use the same locale. For information about configuring locales, see Using the NLS Settings Page of the Management Portal.

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

For an important discussion of performance planning, including memory management and scaling, CPU sizing and scaling, and other considerations, see System Resource Planning and Management.

The most typical distributed cache cluster configuration involves one InterSystems IRIS instance per host, and one cluster role per instance — that is, either data server or application server. When using one of the automated deployment methods described in the next section, this configuration is the only option. The provided procedure for using the Management Portal assumes this configuration as well.

HealthShare Health Connect does not support distributed caching.

2.2.1 Automated Deployment Methods for Clusters

In addition to the manual procedure outlined in this section, InterSystems IRIS Data platform provides two methods of automated deployment of distributed cache clusters that are fully operational following deployment.

2.2.1.1 Deploy a Distributed Cache Cluster Using the InterSystems Kubernetes Operator (IKO)

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 isn't required to deploy InterSystems IRIS under Kubernetes, but it greatly simplifies the process and adds InterSystems IRIS-specific cluster management capabilities to Kubernetes, enabling 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.

2.2.1.2 Deploy a Distributed Cache Cluster Using 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 when restarting 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.

Using configuration merge, you can deploy a distributed cache cluster by calling separate merge files for the data server and the application servers, deploying them sequentially.

The IKO, described above, incorporates the configuration merge feature.

For information about using configuration merge in general and to deploy a distributed cache cluster in particular, see Automating Configuration of InterSystems IRIS with Configuration Merge.

2.2.2 Deploy the Cluster Using the Management Portal

When deploying manually, you must first identify or provision the infrastructure that will host the cluster, then identify or deploy InterSystems IRIS instances on the hosts, and finally configure the deployed instances as a cluster. Once you have deployed or identified the InterSystems IRIS instances you intend to include, and arranged network access of sufficient bandwidth among their hosts, configuring a distributed cache cluster using the Management Portal involves the following steps:

- Prepare the data server
- Configure the application servers

You perform these steps on the ECP Settings page of the Management Portal (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**).

2.2.2.1 Preparing the Data Server

An InterSystems IRIS instance cannot actually operate as the data server in a distributed cache cluster until it is configured as such on the application servers. The procedure for preparing the instance to be a data server, however, includes one required action and two optional actions.

To prepare an instance to be a data server, navigate to the ECP Settings page by selecting **System Administration** on the Management Portal home page, then **Configuration**, then **Connectivity**, then **ECP Settings**, then do the following:

- In the **This System as an ECP Data Server** box on the right, enable the ECP service by clicking the **Enable** link for the service. This opens an Edit Service dialog for %Service_ECP; select **Service Enabled** and click **Save** to enable the service. (If the service is already enabled, as indicated by the presence of a **Disable** link in the box, go on to the next step.)

  Note: For a detailed explanation of InterSystems services, see Services.

- If you want multiple application servers to be able to connect simultaneously to the data server, in the **This System as an ECP Data Server** box, change the **Maximum number of application servers** setting to the number of application servers you want to configure, then click **Save** and restart the instance. (If the number of simultaneous application server connections becomes greater than the number you enter for this setting, the data server instance automatically restarts.)

  Note: The maximum number of application servers can also be set using the MaxServerConn setting in the configuration parameter file (CPF), including in a configuration merge file on UNIX® and Linux platforms.

- The **Time interval for Troubled state** settings determines one of three timeouts used manage recovery of interrupted connections between application servers and the data server; leave it at the default of 60 until you have some data about the cluster's operation over time. For more information on the ECP recovery timeouts, see ECP Recovery Protocol.
- To enable the use of TLS to secure connections from application servers, click the **Set up SSL/TLS '%ECPServer'** link to create an ECP TLS configuration for the data server, then a **ECP SSL/TLS support** setting, as follows:
  - **Required** — An application server can connect only if **Use SSL/TLS** is selected for this data server.
  - **Enabled** — An application server can connect regardless of whether **Use SSL/TLS** is selected for this data server.
  - **Disabled** — An application server cannot connect if **Use SSL/TLS** is selected (default) for this data server.

  As described in Distributed Cache Cluster Security, TLS is one of several options for securing ECP communications. However, enabling TLS may have a significant negative impact on performance. When a cluster's application servers and data server are located in the same data center, which provides optimal performance, the physical security of the data center alone may provide sufficient security for the cluster.

  For important information on enabling and using TLS in a distributed cache cluster, including authorization of secured application server connections on the data server, see Securing Application Server Connections to the Data Server with TLS.

Note: ECP uses some of the database cache on the data server to store various control structures; you may need to increase the size of the database cache or caches to accommodate this. For more information, see Increase Data Server Database Caches for ECP Control Structures.

The data server is now ready to accept connections from valid application servers.

2.2.2.2 Configuring an Application Server

Configuring an InterSystems IRIS instance as an application server in a distributed cache cluster involves two steps:

- Adding the data server instance as a data server on the application server instance.
- Adding the desired databases on the data server as remote databases on the application server.

To add the data server to the application server, do the following:

1. As described for the data server in Preparing the Data Server, navigate to the ECP Settings page; leave the settings on the **This System as an ECP Application Server** side set to the defaults.
2. If the **ECP SSL/TLS support** setting for the data server you are adding is **Enabled** or **Required**, click the **Set up SSL/TLS '%ECPClient'** link to create an ECP TLS configuration for the application server. (You can also do this in the ECP Data Server dialog in a later step.) For more information, see the **Use SSL/TLS** setting in the next step.
3. Click **Data Servers** to display the ECP Data Servers page and click **Add Server**. In the ECP Data Server dialog, enter the following information for the data server:
   - **Server Name** — A descriptive name identifying the data server. (This name is limited to 64 characters.)
   - **Host DNS Name or IP Address** — Specify the DNS name of the data server's host or its IP address (in dotted-decimal format or, if IPv6 is enabled, in colon-separated format). If you use the DNS name, it resolves to an actual IP address each time the application server initiates a connection to that data server host. For more information, see IPv6 Support.

     Important: When adding a mirror primary as a data server (see the **Mirror Connection** setting), do not enter the virtual IP address (VIP) of the mirror, but rather the DNS name or IP address of the current primary failover member.

- **IP Port** — The port number defaults to `1972`, the default InterSystems IRIS superserver (IP) port; change it as necessary to the superserver port of the InterSystems IRIS instance on the data server.
- **Mirror Connection** — Select this check box if the data server is the primary failover member in a mirror. (See Configuring Application Server Connections to a Mirror for important information about configuring a mirror primary as a data server.)
- **Batch Mode** — Select this check box if the server process for this data server should run in batch mode. In batch mode, the data server always loads blocks and caches them in batch level. However, if a block is sent back to the application server, it will be cached regularly.
- **Use SSL/TLS** — Use this check box as follows:
  - If the **ECP SSL/TLS support** setting for the data server you are adding is **Disabled**, it does not matter whether you select this check box; TLS will not be not used to secure connections to the data server.
  - If the **ECP SSL/TLS support** setting for the data server you are adding is **Enabled**, select this check box to use TLS to secure connections to this data server; clear it to not use TLS.
  - If the **ECP SSL/TLS support** setting for the data server you are adding is **Required**, you must select this check box.

  If the **ECP SSL/TLS support** setting for the data server you are adding is **Enabled** or **Required** and you have not yet created a TLS configuration for the application server, click the **Set up SSL/TLS '%ECPClient'** link to do so. For more information on using TLS in a distributed cache cluster, including authorization of secured application server connections on the data server, see Securing Application Server Connections to the Data Server with TLS.

4. Click **Save**. The data server appears in the data server list; you can remove or edit the data server definition, or change its status (see Monitoring Distributed Applications) using the available links. You can also view a list of all application servers connecting to a data server by going to the ECP Settings page on the data server and clicking the **Application Servers** button.

To add each desired database on the data server as a remote database on the application server, you must create a namespace on the application server and map it to that database, as follows:

1. Navigate to the Namespaces page by selecting **System Administration** on the Management Portal home page, then **Configuration**, then **System Configuration**, then **Namespaces**. Click **Create New Namespace** to display the New Namespace page.
2. Enter a name for the new namespace, which typically reflects the name of the remote database it is mapped to.
3. At **The default database for Globals in this namespace is a**, select **Remote Database**, then select **Create New Database** to open the Create Remote Database dialog. In this dialog,
   - Select the data server from the **Remote Server** drop-down.
   - Leave **Remote directory** set to **Select directory from a list** and select the data server database you want to map to the namespace using the **Directory** drop-down, which lists all of the database directories on the data server.
   - Enter a local name for the remote database; this typically reflects the name of the database on the data server, the local name of the data server as specified in the previous procedure, or both.
   - Click **Finish** to add the remote database and map it to the new namespace.
4. At **The default database for Routines in this namespace is a**, select **Remote Database**, then select the database you just created from the drop-down.
5. The namespace does not need to be interoperability-enabled; to save time, you can clear the **Enable namespace for interoperability productions** check box.
6. Select **Save**. The new namespace now appears in the list on the Namespaces list.

Once you have added a data server database as a remote database on the application server, applications can query that database through the namespace it is mapped to on the application server.

Note: Remember that even though a namespace on the application server is mapped to a database on the data server, changes to the namespace mapped to that database *on the data server* are unknown to the application server. (For information about mapping, see Global Mappings.) For example, suppose the namespace DATA on the data server has the default globals database DATA; on the application server, the namespace REMOTEDATA is mapped to the same (remote) database, DATA. If you create a mapping in the DATA namespace on the data server mapping the global **^DATA2** to the DATA2 database, this mapping is not propagated to the application server. Therefore, if you do not add DATA2 as a remote database on the application server and create the same mapping in the REMOTEDATA namespace, queries the application server receives will not be able to read the **^DATA2** global.

2.2.3 Distributed Cache Cluster Security

All InterSystems instances in a distributed cache cluster need to be within the secured InterSystems IRIS perimeter (that is, within an externally secured environment). This is because ECP is a basic security service, rather than a resource-based service, so there is no way to regulate which users have access to it. (For more information on basic and resource-based services, see Available Services.)

However, the following security tools are available:

- Securing application server connections to the data server with TLS
- Restricting incoming access to a data server
- Controlling access to databases with roles and privileges

Note: When databases are encrypted on the data servers, you should also encrypt the IRISTEMP database on all connected application servers. The same or different keys can be used. For more information on database encryption, see Encryption Guide.

2.2.3.1 Securing Application Server Connections to a Data Server with TLS

If TLS is enabled on a data server, you can use it to secure connections from an application server to that data server. This protection includes X.509 certificate-based encryption. For detailed information about TLS and its use with InterSystems products, see InterSystems TLS Guide.

When configuring or editing a data server or at any time thereafter (see Preparing the Data Server), you can select **Enabled** or **Required** as the **ECP SSL/TLS support** setting, rather than the default **Disabled**. These settings control the options for use of the **Use SSL/TLS** check box, which secures connections to a data server with TLS, when adding a data server to an application server (see Configuring an Application Server) or editing an existing data server. These settings have the following effect:

- **Disabled** — The use of TLS for application server connections to this data server is disabled, even for an application server on which **Use SSL/TLS** is selected.
- **Enabled** — The use of TLS for application server connections is enabled on the data server; TLS is used for connections from application servers on which **Use SSL/TLS** is selected, and is not used for connections from application servers on which **Use SSL/TLS** is not selected.
- **Required** — The data server requires application server connections to use TLS; an application server can connect to the data server only if **Use SSL/TLS** is selected for the data server, in which case TLS is used for all connections.

There are three requirements for establishing a connection from an application server to a data server using TLS, as follows:

- The data server must have TLS connections enabled for superserver clients. To do this, on the data server, navigate to the system default superserver configuration page (**System Administration** > **Security** > **Superservers**) and select **Enabled** for the **SSL/TLS support level** setting. See Managing Superservers for more details.
- Both instances must have an ECP TLS configuration.

  For this reason, both sides of the ECP Settings page (**System Administration > Configuration > Connectivity > ECP Settings**) — **This System as an ECP Application Server** and **This System as an ECP Data Server** — include a **Set Up SSL/TLS** link, which you can use to create the appropriate ECP TLS configuration for the instance. To do so, follow this procedure:

  1. On the ECP Settings page, click **Set up SSL/TLS '%ECPClient'** link on the application server side or the **Set up SSL/TLS '%ECPServer'** link on the data server side.
  2. Complete the fields on the form in the **Edit SSL/TLS Configurations for ECP** dialog, These are analogous to those on the **New SSL/TLS Configuration** page, as described in Create or Edit a TLS Configuration. However, there are no **Configuration Name**, **Description**, or **Enabled** fields; also, for the private key password, this page allows you to enter or replace one (**Enter new password**), specify that none is to be used (**Clear password**), or leave an existing one as it is (**Leave as is**).

     Fields on this page are:

     – **File containing trusted Certificate Authority X.509 certificate(s)**

       The path and name of a file that contains the X.509 certificate(s) in PEM format of the Certificate Authority (CA) or Certificate Authorities that this configuration trusts. You can specify either an absolute path or a path relative to the *install-dir*/mgr/ directory. For detailed information about X.509 certificates and their generation and use, see InterSystems TLS Guide.

       Note: This file must include the certificate(s) that can be used to verify the X.509 certificates belonging to other mirror members. If the file includes multiple certificates, they must be in the correct order, as described in Establishing the Required Certificate Chain, with the current instance's certificate first.

     – **File containing this configuration's X.509 certificate**

       The full location of the configuration's own X.509 certificate(s), in PEM format. This can be specified as either an absolute or a relative path.

       Note: The certificate's distinguished name (DN) must appear in the certificate's subject field.

     – **File containing associated private** key

       The full location of the configuration's private key file, specified as either an absolute or relative path.

     – **Private key type**

       The algorithm used to generate the private key, where valid options are **DSA** and **RSA**.

     – **Password**

       Select **Enter new password** when you are creating an ECP TLS configuration, so you can enter and confirm the password for the private key associated with the certificate.

     – **Protocols**

       Those communications protocols that the configuration considers valid; TLSv1.1, and TLSv1.2 are enabled by default.

     – **Enabled ciphersuites**

The set of ciphersuites used to protect communications between the client and the server. Typically you can leave this at the default setting.

Once you complete the form, click **Save**.

- An application server must be authorized on a data server before it can connect using TLS.

  The first time an application server attempts to connect to a data server using TLS, its SSL (TLS) computer name (the Subject Distinguished Name from its X.509 certificate) and the IP address of its host are displayed in a list of **pending ECP application servers to be authorized or rejected** on the data server's Application Servers page (**System Administration > Configuration > Connectivity > ECP Settings > Application Servers**). Use the **Authorize** and **Reject** links to take action on requests in the list. (If there are no pending requests, the list does not display.)

  If one or more application servers have been authorized to connect using TLS, their SSL (TLS) computer names are displayed in a list of **authorized SSL computer names for ECP application servers** on the Application Servers page. You can use the **Delete** link to cancel the authorization. (If there are no authorized application servers, the list does not display.)

2.2.3.2 Restricting Incoming Access to a Data Server

By default, any InterSystems IRIS instance on which the data server instance is configured as a data server (as described in the previous section) can connect to the data server. However, you can restrict which instances can act as application servers for the data server by specifying the hosts from which incoming connections are allowed; if you do this, hosts not explicitly listed cannot connect to the data server. Do this by performing the following steps on the data server:

1. On the Services page (from the portal home page, select **Security** and then **Services**), click **%Service_ECP**. The Edit Service dialog displays.
2. By default, the **Allowed Incoming Connections** box is empty, which means any application server can connect to this instance if the ECP service is enabled; click **Add** and enter a single IP address (such as **192.9.202.55**) or fully-qualified domain name (such as **mycomputer.myorg.com**), or a range of IP addresses (for example,**8.61.202–210.*** or **18.68.*.***). Once there are one or more entries on the list and you click **Save** in the Edit Service dialog, only the hosts specified by those entries can connect.

You can always access the list as described and use a **Delete** to delete the host from the list or an **Edit** link to specify the roles associated with the host, as described in Controlling Access with Roles and Privileges.

2.2.3.3 Controlling Access to Databases with Roles and Privileges

InterSystems uses a security model in which assets, including databases, are assigned to *resources*, and resources are assigned *permissions*, such as read and write. A combination of a resource and a permission is called a *privilege*. Privileges are assigned to *roles*, to which users can belong. In this way, roles are used to control user access to resources. For information about this model, see Authorization: Controlling User Access.

To be granted access to a database on the data server, the role held by the user initiating the process on the application server and the role set for the ECP connection on the data server must both include permissions for the same resource representing that database. For example, if a user belongs to a role on an application server that grants the privilege of read permission for a particular database resource, and the role set for the ECP connection on the data server also includes this privilege, the user can read data from the database on the application server.

By default, InterSystems IRIS grants ECP connections on the data server the **%All** privilege when the data server runs on behalf of an application server. This means that whatever privileges the user on the application server has are matched on the data server, and access is therefore controlled only on the application server. For example, a user on the application server who has privileges only for the **%DB_USER** resource but not the **%DB_IRISLIB** resource can access data in the USER database on the data server, but attempting to access the IRISLIB database on the data server results in a <PROTECT> error. If a different user on the application server has privileges for the **%DB_IRISLIB** resource, the IRISLIB database is available to that user.

Note: InterSystems recommends the use of an LDAP server to implement centralized security. including user roles and privileges, across the application servers of a distributed cache cluster. For information about using LDAP with InterSystems IRIS, see LDAP Guide.

However, you can also restrict the roles available to ECP connections on the data server based on the application server host. For example, on the data server you can specify that when interacting with a specific application server, the only available role is **%DB_USER**. In this case, users on the application server granted the **%DB_USER** role can access the USER database on the data server, but no users on the application server can access any other database on the data server regardless of the roles they are granted.

CAUTION: InterSystems strongly recommends that you secure the cluster by specifying available roles for all application servers in the cluster, rather than allowing the data server to continue to grant the **%All** privilege to all ECP connections.

The following are exceptions to this behavior:

- InterSystems IRIS always grants the data server the **%DB_IRISSYS** role since it requires Read access to the IRISSYS database to run. This means that a user on an application server with **%DB_IRISSYS** can access the IRISSYS database on the data server.

  To prevent a user on the application server from having access to the IRISSYS database on the data server, there are two options:

  – Do not grant the user privileges for the **%DB_IRISSYS** resource.

  – On the data server, change the name of the resource for the IRISSYS database to something other than **%DB_IRISSYS**, making sure that the user on the application server has no privileges for that resource.

- If the data server has any public resources, they are available to any user on the ECP application server, regardless of either the roles held on the application server or the roles configured for the ECP connection.

To specify the available roles for ECP connections from a specific application server on the data server, do the following:

1. Go to the Services page (from the portal home page, select **Security** and then **Services**) and click **%Service_ECP** to display the Edit Service dialog.
2. Click the **Edit** link for the application server host you want to restrict to display the **Select Roles** area.
3. To specify roles for the host, select roles from those listed under **Available** and click the right arrow to add them to the **Selected** list.
4. To remove roles from the **Selected** list, select them and then click the left arrow.
5. To add all roles to the **Selected** list, click the double right arrow; to remove all roles from the **Selected** list, click the double left arrow.
6. Click **Save** to associate the roles with the IP address.

By default, a listed host holds the **%All** role, but if you specify one or more other roles, these roles are the only roles that the connection holds. Therefore, a connection from a host or IP range with the **%Operator** role has only the privileges associated with that role, while a connection from a host with no associated roles (and therefore **%All**) has all privileges.

Changes to the roles available to application server hosts and to the public permissions on resources on the data server require a restart of InterSystems IRIS before taking effect.

2.2.3.4 Security-Related Error Reporting

The behavior of security-related error reporting with ECP varies depending on whether the check fails on the application server or the data server and the type of operation:

- If the check fails on the application server, there is an immediate <PROTECT> error.
- For synchronous operations on the data server, there is an immediate <PROTECT> error.
- For asynchronous operations on the data server, there is a possibly delayed <NETWORK DATA UPDATE FAILED> error. This includes **Set** operations.

2.3 Monitoring Distributed Cache Applications

A running distributed cache cluster consists of a data server instance — a data provider — connected to one or more application server systems—data consumers. Between each application server and the data server, there is an *ECP connection* — a TCP/IP connection that ECP uses to send data and commands.

You can monitor the status of the servers and connections in a distributed cache cluster on the ECP Settings page (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**).

The **ECP Settings** page has two subsections:

1. **This System as an ECP Data Server** displays settings for the data server as well as the status of the ECP service.
2. **This System as an ECP Application Server** displays settings for the application server.

The following sections describe status information for connections:

- ECP Connection Information
- ECP Connection States
- ECP Connection Operations

2.3.1 ECP Connection Information

Click the **Data Servers** button on the ECP Data Servers Settings page (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**) to display the ECP Data Servers page, which lists the current data server connections on the application server. The ECP Application Servers page, which you can display by clicking the **Application Servers** button on the ECP Settings page, contains a list of the current application server connections on the data server.

2.3.1.1 Data Server Connections

The ECP Data Servers page displays the following information for each *data server* connection:

Server Name

The logical name of the data server system on this connection, as entered when the server was added to the application server configuration.

Host Name

The host name of the data server system, as entered when the server was added to the application server configuration.

IP Port

The IP port number used to connect to the data server.

Status

The current status of this connection. Connection states are described in ECP Connection States.

Edit

If the current status of this connection is **Not Connected** or **Disabled**, you can edit the port and host information of the data server.

Change Status

From each data server row you can change the status of an existing ECP connection with that data server; see ECP Connection Operations for more information.

Delete

You can delete the data server information from the application server.

2.3.1.2 Application Server Connections

Click **ECP Application Servers** on the ECP Settings page (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**) to view the ECP Application Servers page with a list of application server connections on this data server:

Client Name

The logical name of the application server on this connection.

Status

The current status of this connection. Connection states are described in ECP Connection States.

Client IP

The host name or IP address of the application server

IP Port

The port number used to connect to the application server.

2.3.2 ECP Connection States

In an operating cluster, an ECP connection can be in one of the following states:

Table 2–1: ECP Connection States

State	Description
Not Connected	The connection is defined but has not been used yet.
Connection in Progress	The connection is in the process of establishing itself. This is a transitional state that lasts only until the connection is established.
Normal	The connection is operating normally and has been used recently.
Trouble	The connection has encountered a problem. If possible, the connection automatically corrects itself.
Disabled	The connection has been manually disabled by a system administrator. Any application making use of this connection receives a `<NETWORK>` error.

The following sections describe each connection state as it relates to application servers or the data server:

- Application Server Connection States
- Data Server Connection States

2.3.2.1 Application Server Connection States

The following entries describe the application server side of each of the connection states. The numerical values provided allow you to determine the connection state indicated in a log message; for example, the following message refers to the **`Application Server Trouble State`**: jojo96HABER

```
01/28/24-00:00:11:859 (6552) 2
[SYSTEM MONITOR]
ECPClientState Alert: ECP reports Clients state 6
```

- Not Initialized (0)
- Not Connected (1)
- Connection in Progress (2)
- Connection Failed (3)
- Disabled (4)
- Normal (5)
- Trouble (6)
- Transitional Recovery (7)

Application Server Not Initiatized (0)

The node is in the state of being initialized (very rare), or has not yet been initialized.

Application Server Not Connected State (1)

An application server-side ECP connection starts out in the *Not Connected* state. In this state, there are no ECP daemons for the connection. If an application server process makes a network request, daemons are created for the connection and the connection enters the *Connection in Progress* state.

Application Server Connection in Progress State (2)

In the *Connection in Progress* state, a network daemon exists for the connection and actively tries to establish a connection to the data server; when the connection is established, it enters the *Normal* state. While the connection is in the *Connection in Progress* state, the user process must wait for up to 20 seconds for it to be established. If the connection is not established within that time, the user process receives a `<NETWORK>` error.

The application server ECP daemon attempts to create a new connection to the data server in the background. If no connection is established within 20 minutes, the connection returns to the *Not Connected* state and the daemon for the connection goes away.

Application Server Connection Failed State (3)

A connection attempt while in the *Connection in Progress* state failed. This state persists for a few seconds before transitioning to the *Application Server Not Connected* state.

Application Server Disabled State (4)

An ECP connection is marked *Disabled* if an administrator declares that it is disabled. In this state, no daemons exist and any network requests that would use that connection immediately receive `<NETWORK>` errors.

Application Server Normal State (5)

After a connection completes, it enters the *Normal* (data transfer) state. In this state, the application server-side daemons exist and actively send requests and receive answers across the network. The connection stays in the *Normal* state until the connection becomes unworkable or until the application server or the data server requests a shutdown of the connection.

Application Server Trouble State (6)

If the connection from the application server to the data server encounters problems, the application server ECP connection enters the *Trouble* state. In this state, application server ECP daemons exist and are actively try to restore the connection. An underlying TCP connection may or may not still exist. The recovery method is similar whether or not the underlying TCP connection gets reset and must be recreated, or if it stops working temporarily.

During the application server **Time to wait for recovery** timeout (default of 20 minutes), the application server attempts to reconnect to the data server to perform ECP connection recovery. During this interval, existing network requests are preserved, but the originating application server-side user process blocks new network requests, waiting for the connection to resume. If the connection returns within the **Time to wait for recovery** timeout, it returns to the *Normal* state and the blocked network requests proceed.

For example, if a data server goes offline, any application server connected to it has its state set to *Trouble* until the data server becomes available. If the problem is corrected gracefully, a connection's state reverts to *Normal*; otherwise, if the trouble state is not recovered, it reverts to *Not Connected*.

Applications continue running until they require network access. All locally cached data is available to the application while the server is not responding.

Application Server Transitional Recovery States (7)

Transitional recovery states are part of the *Trouble* state. If there is no current TCP connection to the data server, and a new connection is established, the application server and data server engage in a *recovery protocol* which flushes the application server cache, recovers transactions and locks, and returns to the *Normal* state.

Similarly, if the data server shuts down, either gracefully or as a result of a crash, and then restarts, it enters a short period (approximately 30 seconds) during which it allows application servers to reconnect and recover their existing sessions. Once again, the application server and the data server engage in the recovery protocol.

If connection recovery is not complete within the **Time to wait for recovery** timeout, the application server gives up on connection recovery. Specifically, the application server returns errors to all pending network requests and changes the connection state to *Not Connected*. If it has not already done so, the data server rolls back all the transactions and releases all the locks from this application server the next time this application server connects to the data server.

If the recovery is successful, the connection returns to the *Normal* state and the blocked network requests proceed.

2.3.2.2 Data Server Connection States

The following sections describe the data server side of each of the connection states:

- Free
- Normal
- Trouble

- Recovering

Data Server Free States

When an ECP server instance starts up, all incoming ECP connections are in an initial "unassigned" *Free* state and are available for connections from any application server that is listed in the connection access control list. If a connection from an application server previously existed and has since gone away, but does not require any recovery steps, the connection is placed in the "idle" *Free* state. The only difference between these two states is that in the idle state, this connection block is already assigned to a particular application server, rather than being available for any application server that passes the access control list.

Data Server Normal State

In the data server *Normal* state, the application server connection is normal. At any point in the processing of incoming connections, whenever the application server disconnects from the data server (except as part of the data server's own shutdown sequence), the data server rolls back any pending transactions and releases any incoming locks from that application server, and places the application server connection in the "idle" *Free* state.

Data Server Trouble States

If the application server is not responding, the data server shows a *Trouble* state. If the data server crashes or shuts down, it remembers the connections that were active at the time of the crash or shutdown. After restarting, the data server waits for a brief time (usually 30 seconds) for application servers to reclaim their sessions (locks and open transactions). If an application server does not complete recovery during this awaiting recovery interval, all pending work on that connection is rolled back and the connection is placed in the "idle" state.

Data Server Recovering State

The data server connection is in a recovery state for a very short time when the application server is in the process of reclaiming its session. The data server keeps the application server in trouble state for the **Time interval for Troubled state** timeout (default is 60 seconds) for it to reclaim the connection; otherwise, it releases the application resources (rolls back all open transactions and releases locks) and then sets the state to *Free*.

2.3.3 ECP Connection Operations

On the ECP Data Servers page (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**, click **Data Servers** button) on an application server, you can change the status of the ECP connection. In each data server row, click **Change Status** to display the connection information and perform the appropriate selection of the following choices:

Change to Disabled

Set the state of this connection to *Disabled*. This releases any locks held for the application server, rolls back any open transactions involving this connection, and purges cached blocks from the data server. If this is an active connection, the change in status sends an error to all applications waiting for network replies from the data server.

Change to Normal

Set the state of this connection to *Normal*.

Change to Not Connected

Set the state of this connection to *Not Connected*. As with changing the state to **Disabled**, this releases any locks held for the application server, rolls back any open transactions involving this connection, and purges cached blocks from the data server. If this is an active connection, the change in status sends an error to all applications waiting for network replies from the data server.

2.4 Developing Distributed Cache Applications

This section discusses application development and design issues that are helpful if you would like to deploy your application on a distributed cache cluster, either as an option or as its primary configuration.

With InterSystems IRIS, the decision to deploy an application as a distributed system is primarily a runtime configuration issue (see Deploying a Distributed Cache Cluster). Using InterSystems IRIS configuration tools, map the logical names of your data (globals) and application logic (routines) to physical storage on the appropriate system.

This section discusses the following topics:

- ECP Recovery Protocol
- Forced Disconnects
- Performance and Programming Considerations
- ECP-related Errors

2.4.1 ECP Recovery Protocol

ECP is designed to automatically recover from interruptions in connectivity between an application server and the data server. In the event of such an interruption, ECP executes a recovery protocol that differs depending on the nature of the failure. The result is that the connection is either recovered, allowing the application processes to continue as though nothing had happened, or reset, forcing transaction rollback and rebuilding of the application processes. The main principles are as follows:

- When the connection between an application server and data server is interrupted, the application server attempts to reestablish its connection with the data server, repeatedly if necessary, at an interval determined by the **Time between reconnections** setting (5 seconds by default).
- When the interruption is brief, the connection is recovered.

  If the connection is reestablished within the data server's configured **Time interval for Troubled state** timeout period (60 seconds by default), the data server restores all locks and open transactions to the state they were in prior to the interruption.
- If the interruption is longer, the data server resets the connection, so that it cannot be recovered when the interruption ends.

  If the connection is not reestablished within the **Time interval for Troubled state**, the data server unilaterally resets the connection, allowing it to roll back transactions and release locks from the unresponsive application server so as not to block functioning application servers. When connectivity is restored, the connection is disabled from the application server point of view; all processes waiting for the data server on the interrupted connection receive a **<NETWORK>** error and enter a rollback-only condition. The next request received by the application server establishes a new connection to the data server.
- If the interruption is very long, the application server also resets the connection.

  If the connection is not reestablished within the application server's longer Time to wait for recovery timeout period (20 minutes by default), the application server unilaterally resets the connection; all processes waiting for the data server on the interrupted connection receive a **<NETWORK>** error and enter a rollback-only condition. The next request received by the application server establishes a new connection to the data server, if possible.

The ECP timeout settings are shown in the following table. Each is configurable on the **System** > **Configuration** > **ECP Settings** page of the Management Portal, or in the ECP section of in the configuration parameter file (CPF); for more information, see ECP.

Table 2–2: ECP Timeout Values

Management Portal Setting	CPF Setting	Default	Range	
Time between reconnections	ClientReconnectInterval	5 seconds	1–60 seconds	The interval at which an application makes attempts to reconnect to the data server.
Time interval for Troubled state	ServerTroubleDuration	60 seconds	20–65535 seconds	The length of time for which the data server waits for contact from the application server before resetting an interrupted connection.
Time to wait for recovery	ClientReconnectDuration	1200 seconds (20 minutes)	10–65535 seconds	The length of time for which an application server continues attempting to reconnect to the data server before resetting an interrupted connection.

The default values are intended to do the following:

- Avoid tying up data server resources that could be used for other application servers for a long time by waiting for an application server to become available.
- Give an application server — which has nothing else to do when the data server is not available — the ability to wait out an extended connection interruption for much longer by trying to reconnect at frequent intervals.

ECP relies on the TCP physical connection to detect the health of the instance at the other end without using too much of its capacity. On most platforms, you can adjust the TCP connection failure and detection behavior at the system level.

While an application server connection becomes inactive, the data server maintains an active daemon waiting for new requests to arrive on the connection, or for a new connection to be requested by the application server. If the old connection returns, it can immediately resume operation without recovery. When the underlying heartbeat mechanism indicates that the application server is completely unavailable due to a system or network failure, the underlying TCP connection is quickly reset. Thus, an extended period without a response from an application server generally indicates some kind of problem on the application server that caused it to stop functioning, but without interfering with its connections.

If the underlying TCP connection is reset, the data server puts the connection in an "awaiting reconnection" state in which there is no active ECP daemon on the data server. A new pair of data server daemons are created when the next incoming connection is requested by the application server.

Collectively, the nonresponsive state and the awaiting reconnection state are known as the data server *Trouble* state. The recovery required in both cases is very similar.

If the data server fails or shuts down, it remembers the connections that were active at the time of the crash or shutdown. After restarting, the data server has a short window (usually 30 seconds) during which it places these interrupted connections in the awaiting reconnection state. In this state, the application server and data server can cooperate together to recover all the transaction and lock states as well as all the pending **Set** and **Kill** transactions from the moment of the data server shutdown.

During the recovery of an ECP-configured instance, InterSystems IRIS guarantees a number of recoverable semantics, and also specifies limitations to these guarantees. ECP Recovery Process, Guarantees, and Limitations describes these in detail, as well as providing additional details about the recovery process.

2.4.2 Forced Disconnects

By default, ECP automatically manages the connection between an application server and a data server. When an ECP-configured instance starts up, all connections between application servers and data servers are in the *Not Connected* state (that is, the connection is defined, but not yet established). As soon as an application server makes a request (for data or

code) that requires a connection to the data server, the connection is automatically established and the state changes to *Normal*. The network connection between the application server and data server is kept open indefinitely.

In some applications, you may wish to close open ECP connections. For example, suppose you have a system, configured as an application server, that periodically (a few times a day) needs to fetch data stored on a data server system, but does not need to keep the network connection with the data server open afterwards. In this case, the application server system can issue a call to the **SYS.ECP.ChangeToNotConnected()** method to force the state of the ECP connection to *Not Connected*.

For example:

ObjectScript

```
Do OperationThatUsesECP()
Do SYS.ECP.ChangeToNotConnected("ConnectionName")
```

The **ChangeToNotConnected** method does the following:

1. Completes sending any data modifications to the data server and waits for acknowledgment from the data server.
2. Removes any locks on the data server that were opened by the application server.
3. Rolls back the data server side of any open transactions. The application server side of the transaction goes into a "rollback only" condition.
4. Completes pending requests with a <NETWORK> error.
5. Flushes all cached blocks.

After completion of the state change to *Not Connected*, the next request for data from the data server automatically reestablishes the connection.

Note: See Data Server Connections for information about changing data server connection status from the Management Portal.

2.4.3 Performance and Programming Considerations

To achieve the highest performance and reliability from distributed cache cluster-based applications, you should be aware of the following issues:

- Do Not Use Multiple ECP Channels
- Increase Data Server Database Caches for ECP Control Structures
- Evaluate the Effects of Load Balancing User Requests
- Restrict Transactions to a Single Data Server
- Locate Temporary Globals on the Application Server
- Avoid Repeated References to Undefined Globals
- Avoid the Use of Stream Fields
- The $Increment Function and Application Counters

2.4.3.1 Do Not Use Multiple ECP Channels

InterSystems strongly discourages establishing multiple duplicate ECP channels between an application server and a data server to try to increase bandwidth. You run the dangerous risk of having locks and updates for a single logical transaction arrive out-of-sync on the data server, which may result in data inconsistency.

2.4.3.2 Increase Data Server Database Caches for ECP Control Structures

In addition to buffering the blocks that are served over ECP, data servers use global buffers to store various ECP control structures. There are several factors that go into determining how much memory these structures might require, but the most significant is a function of the aggregate sizes of the clients' caches. To roughly approximate the requirements, so you can adjust the data server's database caches if needed, use the following guidelines:

Database Block Size	Recommendation
8 KB	50 MB plus 1% of the sum of the sizes of all of the application servers' 8 KB database caches
16 KB (if enabled)	0.5% of the sum of the sizes of all of the application servers' 16 KB database caches
32 KB (if enabled)	0.25% of the sum of the sizes of all of the application servers' 32 KB database caches
64 KB (if enabled)	0.125% of the sum of the sizes of all of the application servers' 64 KB database caches

For example, if the 16 KB block size is enabled in addition to the default 8 KB block size, and there are six application servers, each with an 8 KB database cache of 2 GB and a 16 KB database cache of 4 GB, you should adjust the data server's 8 KB database cache to ensure that 52 MB (50MB + [12 GB * .01]) is available for control structures, and the 16 KB cache to ensure that 2 MB (24 GB * .005) is available for control structures (rounding up in both cases).

For information about allocating memory to database caches, see Memory and Startup Settings.

2.4.3.3 Evaluate the Effects of Load Balancing User Requests

Connecting users to application servers in a round-robin or load balancing scheme may diminish the benefit of caching on the application server. This is particularly likely if users work in functional groups that have a tendency to read the same data. As these users are spread among application servers, each application server may end up requesting exactly the same data from the data server, which not only diminishes the efficiency of distributed caching using multiple caches for the same data, but can also lead to increased block invalidation as blocks are modified on one application server and refreshed across other application servers. This is somewhat subjective, but someone very familiar with the application characteristics should consider this possible condition. If you do decide to configure a load balancer, see Load Balancing, Failover, and Mirrored Configurations for an important discussion of load balancing a web server tier distributing application connections across application servers.

2.4.3.4 Restrict Transactions to a Single Data Server

Restrict updates within a single transaction to either a single remote data server or the local server. When a transaction includes updates to more than one server (including the local server) and the **TCommit** cannot complete successfully, some servers that are part of the transaction may have committed the updates while others may have rolled them back. For details, see Commit Guarantee.

Note: Updates to IRISTEMP are not considered part of the transaction for the purpose of rollback, and, as such, are not included in this restriction.

2.4.3.5 Locate Temporary Globals on the Application Server

Temporary (scratch) globals should be local to the application server, assuming they do not contain data that needs to be globally shared. Often, temporary globals are highly active and write-intensive. If temporary globals are located on the data server, this may penalize other application servers sharing the ECP connection.

2.4.3.6 Avoid Repeated References to Undefined Globals

Repeated references to a global that is not defined (for example, **$Data(^x(1))** where ^*x* is not defined) always requires a network operation to test if the global is defined on the data server.

By contrast, repeated references to undefined nodes within a defined global (for example, **$Data(^x(1))** where any other node in ^*x* is defined) does not require a network operation once the relevant portion of the global (^*x*) is in the application server cache.

This behavior differs significantly from that of a non-networked application. With local data, repeated references to the undefined global are highly optimized to avoid unnecessary work. Designers porting an application to a networked environment may wish to review the use of globals that are sometimes defined and sometimes not. Often it is sufficient to make sure that some other node of the global is always defined.

2.4.3.7 Avoid the Use of Stream Fields

A stream field within a query results in a read lock, which requires a connection to the data server. For this reason, such queries do not benefit from the database cache, which means their performance does not improve on the second and subsequent invocation.

2.4.3.8 Use the $Increment Function for Application Counters

A common operation in online transaction processing systems is generating a series of unique values for use as record numbers or the like. In a typical relational application, this is done by defining a table that contains a "next available" counter value. When the application needs a new identifier, it locks the row containing the counter, increments the counter value, and releases the lock. Even on a single-server system, this becomes a concurrency bottleneck: application processes spend more and more time waiting for the locks on this common counter to be released. In a networked environment, it is even more of a bottleneck at some point.

InterSystems IRIS addresses this by providing the **$Increment** function, which automatically increments a counter value (stored in a global) without any need of application-level locking. Concurrency for **$Increment** is built into the InterSystems IRIS database engine as well as ECP, making it very efficient for use in single-server as well as in distributed applications.

Applications built using the default structures provided by InterSystems IRIS objects (or SQL) automatically use **$Increment** to allocate object identifier values. **$Increment** is a synchronous operation involving journal synchronization when executed over ECP. For this reason, **$Increment** over ECP is a relatively slow operation, especially compared to others which may or may not already have data cached (in either the application server database cache or the data server database cache). The impact of this may be even greater in a mirrored environment due to network latency between the failover members. For this reason, it may be useful to redesign an application to replace **$Increment** with the **$Sequence** function, which automatically assigns batches of new values to each process on each application server, involving the data server only when a new batch of values is needed. (This approach cannot be used, however, when consecutive application counter values are required.) **$Sequence** can also be used in combination with **$Increment**.

2.4.4 ECP-related Errors

There are several runtime errors that can occur on a system using ECP. An ECP-related error may occur immediately after a command is executed or, in the case of commands that are asynchronous in nature, such as **Kill**, the error occurs a short time after the command completes.

2.4.4.1 <NETWORK> Errors

A <NETWORK> error indicates an error that could not be handled by the normal ECP recovery mechanism.

In an application, it is always acceptable to halt a process or roll back any pending work whenever a <NETWORK> error is received. Some <NETWORK> errors are essentially fatal error conditions. Others indicate a temporary condition that might

clear up soon; however, the expected programming practice is to always roll back any pending work in response to a <NETWORK> error and start the current transaction over from the beginning.

A <NETWORK> error on a get-type request such as **$Data** or **$Order** can often be retried manually rather than simply rolling back the transaction immediately. ECP tries to avoid giving a <NETWORK> error that would lose data, but gives an error more freely for requests that are read-only.

2.4.4.2 Rollback Only Condition

The application-side *rollback-only* condition occurs when the data server detects a network failure during a transaction initiated by the application server and enters a state in which all network requests are met with errors until the transaction is rolled back.

2.5 ECP Recovery Process, Guarantees, and Limitations

The ECP recovery protocol is summarized in ECP Recovery Protocol. This section describes ECP recovery in detail, including its guarantees and limitations.

The simplest case of ECP recovery is a temporary network interruption that is long enough to be noticed, but short enough that the underlying TCP connection stays active during the outage. During the outage, the application server notices that the connection is nonresponsive and blocks new network requests for that connection. Once the connection resumes, processes that were blocked are able to send their pending requests.

If the underlying TCP connection is reset, the data server waits for a reconnection for the **Time interval for Troubled state** setting (one minute by default). If the application server does not succeed in reconnecting during that interval, the data server resets its connection, rolls back its open transactions, and releases its locks. Any subsequent connection from that application server is converted into a request for a brand new connection and the application server is notified that its connection is reset.

The application server keeps a queue of locks to remove and transactions to roll back once the connection is reestablished. By keeping this queue, processes on the application server can always halt, whether or not the data server on which it has pending transactions and locks is currently available. ECP recovery completes any pending Set and Kill operations that had been queued for the data server before the network outage was detected, before it completes the release of locks.

Any time a data server learns that an application server has reset its own connection (due to application server restart, for example), even if it is still within the **Time interval for Troubled state**, the data server resets the connection immediately, rolling back transactions and releasing locks on behalf of that application server. Since the application server's state was reset, there is no longer any state to be maintained by the data server on its behalf.

The final case is when the data server shut down, either gracefully or as a result of a crash. The application server maintains the application state and tries to reconnect to the data server for the **Time to wait for recovery** setting (20 minutes by default). The data server remembers the application server connections that were active at the time of the crash or shutdown; after restarting, it waits up to thirty seconds for those application servers to reconnect and recover their connections. Recovery involves several steps on the data server, some of which involve the data server journal file in very significant ways. The result of the several different steps is that:

- The data server's view of the current active transactions from each application server has been restored from the data server's journal file.
- The data server's view of the current active **Lock** operations from each application server has been restored, by having the application server upload those locks to the data server.
- The application server and the data server agree on exactly which requests from the application server can be ignored (because it is certain they completed before the crash) and which ones should be replayed. Therefore, the last recovery step is to simply let the pending network requests complete, but only those network requests that are safe to replay.

- Finally, the application server delivers to the data server any pending unlock or rollback indications that it saved from jobs that halted while the data server was restarting. All guarantees are maintained, even in the face of sudden and unanticipated data server crashes, as long as the integrity of the storage devices (for database, WIJ, and journal files) are maintained.

During the recovery of an ECP-configured system, InterSystems IRIS guarantees a number of recoverable semantics which are described in detail in ECP Recovery Guarantees. Limitations to these guarantees are described in detail in ECP Recovery Limitations.

2.5.1 ECP Recovery Guarantees

During the recovery of an ECP-configured system, InterSystems IRIS guarantees the following recoverable semantics:

- In-order Updates Guarantee
- ECP Lock Guarantee
- Clusters Lock Guarantee
- Rollback Guarantee
- Commit Guarantee
- Transactions and Locks Guarantee
- ECP Rollback Only Guarantee
- ECP Transaction Recovery Guarantee
- ECP Lock Recovery Guarantee
- $Increment Ordering Guarantee
- ECP Sync Method Guarantee

In the description of each guarantee the first paragraph describes a specific condition. Subsequent paragraphs describe the data guarantee applicable to that particular situation.

In these descriptions, Process A, Process B and so on refer to processes attempting update globals on a data server. These processes may originate on the same or different application servers, or on the data server itself; in some cases the origins of processes are specified, in others they are not germane.

2.5.1.1 In-order Updates Guarantee

Process A updates two data elements sequentially, first global ^*x* and next global ^*y*, where ^*x* and ^*y* are located on the same data server.

If *Process B* sees the change to ^*y*, it also sees the change to ^*x*. This guarantee applies whether or not *Process A* and *Process B* are on the same application server as long as the two data items are on the same data server and the data server remains up.

Process B's ability to view the data modified by *Process A* does not ensure that **Set** operations from *Process B* are restored after the **Set** operations from *Process A*. Only a **Lock** or a **$Increment** operation can ensure proper ordering of competing **Set** and **Kill** operations from two different processes during cluster failover or cluster recovery.

See the Loose Ordering in Cluster Failover or Restore limitation regarding the order in which competing **Set** and **Kill** operations from separate processes are applied during cluster dejournaling and cluster failover.

Important: This guarantee does not apply if the data server crashes, even if ^*x* and ^*y* are journaled. See the Dirty Data Reads for ECP Without Locking limitation for a case in which processes that fit this description can see dirty data that never becomes durable before the data server crash.

2.5.1.2 ECP Lock Guarantee

Process B on DataServer S acquires a lock on global ^*x*, which was once locked by *Process A*.

Process B can see *all* updates on DataServer S done by *Process A* (while holding a lock on ^*x*). Also, if *Process C* sees the updates done by *Process B* on DataServer S (while holding a lock on ^*x*), *Process C* is guaranteed to also see the updates done by *Process A* on DataServer S (while holding a lock on ^*x*).

Serializability is guaranteed whether or not *Process A*, *Process B*, and *Process C* are located on the same application server or on DataServer S itself, as long as DataServer S stays up throughout.

Important: The lock and the data it protects must reside on the same data server.

2.5.1.3 Clusters Lock Guarantee

Process B on a cluster member acquires a lock on global ^*x* in a clustered database; a lock once held by *Process A*.

Process B sees *all* updates to any clustered database done by *Process A* (while holding a lock on ^*x*).

Additionally, if *Process C* on a cluster member sees the updates on a clustered database made by *Process B* (while holding a lock on ^*x*), *Process C* also sees the updates made by *Process A* on any clustered database (while holding a lock on ^*x*).

Serializability is guaranteed whether or not *Process A*, *Process B*, and *Process C* are located on the same cluster member, and whether or not any cluster member crashes.

Important: See the Dirty Data Reads When Cluster Member Crashes limitation regarding transactions on one cluster member seeing dirty data from a transaction on a cluster member that crashes.

2.5.1.4 Rollback Guarantee

Process A executes a **TStart** command, followed by a series of updates, and either halts before issuing a **TCommit**, or executes a **TRollback** before executing a **TCommit**.

All the updates made by *Process A* as part of the transaction are rolled back in the reverse order in which they originally occurred.

Important: See the rollback-related limitations: Conflicting, Non-Locked Change Breaks Rollback, Journal Discontinuity Breaks Rollback, and Asynchronous TCommit Converts to Rollback for more information.

2.5.1.5 Commit Guarantee

Process A makes a series of updates on DataServer S and halts after starting the execution of a **TCommit**.

On each DataServer S that is part of the transaction, the data modifications on DataServer S are either committed or rolled back. If the process that executes the **TCommit** has the **Perform Synchronous Commit** property turned on (`SynchCommit=1`, in the configuration file) and the **TCommit** operation returns without errors, the transaction is guaranteed to have durably committed on all the data servers that are part of the transaction.

Important: If the transaction includes updates to more than one server (including the local server) and the **TCommit** cannot complete successfully, some servers that are part of the transaction may have committed the updates while others may have rolled them back.

2.5.1.6 Transactions and Locks Guarantee

Process A executes a **TStart** for *Transaction T*, locks global ^*x* on DataServer S, and unlocks ^*x* (unlock does not specify the "immediate unlock" lock type).

InterSystems IRIS guarantees that the lock on ^*x* is not released until the transaction has been either committed or rolled back. No other process can acquire a lock on ^*x* until *Transaction T* either commits or rolls back on DataServer S.

Once *Transaction T* commits on DataServer S, *Process B* that acquires a lock on ^*x* sees changes on DataServer S made by *Process A* during *Transaction T*. Any other process that sees changes on DataServer S made by *Process B* (while holding a lock on ^*x*) sees changes on DataServer S made by *Process A* (while executing *Transaction T*). Conversely, if *Transaction T* rolled back on DataServer S, a *Process B* that acquires a lock on ^*x*, sees *none* of the changes made by *Process A* on DataServer S.

Important: See the Conflicting, Non-Locked Change Breaks Rollback limitation for more information.

2.5.1.7 ECP Rollback Only Guarantee

Process A on AppServer C makes changes on DataServer S that are part of a *Transaction T*, and DataServer S unilaterally rolls back those changes (which can happen in certain network outages or data server outages).

All subsequent network requests to DataServer S by *Process A* are rejected with `<NETWORK>` errors until *Process A* explicitly executes a **TRollback** command.

Additionally, if any process on AppServer C completes a network request to DataServer S between the rollback on DataServer S and the **TCommit** of *Transaction T* (AppServer C finds out about the rollback-only condition before the **TCommit**), *Transaction T* is guaranteed to roll back on *all* data servers that are part of *Transaction T*.

2.5.1.8 ECP Transaction Recovery Guarantee

An data server crashes in the middle of an application server transaction, restarts, and completes recovery within the application server recovery timeout interval.

The transaction can be completed normally without violating any of the described guarantees. The data server does not perform any data operations that violate the ordering constraints defined by lock semantics. The only exception is the **$Increment** function (see ECP and Clusters $Increment Limitation for more information). Any transactions that cannot be recovered are rolled back in a way that preserves lock semantics.

Important: InterSystems IRIS expects but does not guarantee that in the absence of continuing faults (whether in the network, the data server, or the application server hardware or software), all or most of the transactions pending into a data server at the time of a data server outage are recovered.

2.5.1.9 ECP Lock Recovery Guarantee

DataServer S has an unplanned shutdown, restarts, and completes recovery within the recovery interval.

The ECP Lock Guarantee still applies as long as all the modified data is journaled. If data is not being journaled, updates made to the data server before the crash can disappear without notice to the application server. InterSystems IRIS no longer guarantees that a process that acquires the lock sees all the updates that were made earlier by other processes while holding the lock.

If DataServer S shuts down gracefully, restarts, and completes recovery within the recovery interval, the ECP Lock Guarantee still applies whether or not data is being journaled.

Updates that are part of a transaction are always journaled; the ECP Transaction Recovery Guarantee applies in a stronger form. Other updates may or may not be journaled, depending on whether or not the destination global in the destination database is marked for journaling.

2.5.1.10 $Increment Ordering Guarantee

The **$Increment** function induces a loose ordering on a series of **Set** and **Kill** operations from separate processes, even if those operations are not protected by a lock.

For example: *Process A* performs some **Set** and **Kill** operations on DataServer S and performs a **$Increment** operation to a global ^*x* on DataServer S. *Process B* performs a subsequent **$Increment** to the same global ^*x*. Any process, including *Process B*, that sees the result of *Process B* incrementing ^*x*, sees all changes on DataServer S that *Process A* made before incrementing ^*x*.

Important: See ECP and Clusters $Increment Limitation for more information.

2.5.1.11 ECP Sync Method Guarantee

Process A updates a global located on Data Server S, and issues a **$system.ECP.Sync()** call to S. Process B then issues a **$system.ECP.Sync()** to S. Process B can see all updates performed by Process A on Data Server S prior to its **$system.ECP.Sync()** call.

$system.ECP.Sync() is relevant only for processes running on an application server. If either process A or B are running on DataServer S itself, then that process does not need to issue a **$system.ECP.Sync()**. If both are running on DataServer S then neither needs **$system.ECP.Sync**, and this is simply the statement that global updates are immediately visible to processes running on the same server.

Important: **$system.ECP.Sync()** does not guarantee durability; see the Dirty Data Reads in ECP without Locking limitation.

2.5.2 ECP Recovery Limitations

During the recovery of an ECP-configured system, there are the following limitations to the InterSystems IRIS guarantees:

- ECP and Clusters $Increment Limitation
- ECP Cache Liveness Limitation
- ECP Routine Revalidation Limitation
- Conflicting, Non-Locked Change Breaks Rollback
- Journal Discontinuity Breaks Rollback
- ECP Can Miss Error After Recovery
- Partial Set or Kill Leads to Journal Mismatch
- Loose Ordering in Cluster Failover or Restore
- Dirty Data Reads When Cluster Member Crashes
- Dirty Data Reads in ECP without Locking
- Asynchronous TCommit Converts to Rollback

2.5.2.1 ECP and Clusters $Increment Limitation

If a data server crashes while the application server has a **$Increment** request outstanding to the data server and the global is journaled, InterSystems IRIS attempts to recover the **$Increment** results from the journal; it does not re-increment the reference.

2.5.2.2 ECP Cache Liveness Limitation

In the absence of continuing faults, application servers observe data that is no more than a few seconds out of date, but this is not guaranteed. Specifically, if an ECP connection to the data server becomes nonfunctional (network problems, data server shutdown, data server backup operation, and so on), the user process may observe data that is arbitrarily stale, up to an application server connection-timeout value. To ensure that data is not stale, use the **Lock** command around the data-

fetch operation, or use **$system.ECP.Sync**. Any network request that makes a round trip to the data server updates the contents of the application server ECP network cache.

2.5.2.3 ECP Routine Revalidation Limitation

If an application server downloads routines from a data server and the data server restarts (planned or unplanned), the routines downloaded from the data server are marked as if they had been edited.

Additionally, if the connection to the data server suffers a network outage (neither application server nor data server shuts down), the routines downloaded from the data server are marked as if they had been edited. In some cases, this behavior causes spurious `<EDITED>` errors as well as `<ERRTRAP>` errors.

2.5.2.4 Conflicting, Non-Locked Change Breaks Rollback

In InterSystems IRIS, the **Lock** command is only advisory. If *Process A* starts a transaction that is updating global ^*x* under protection of a lock on global ^*y*, and another process modifies ^*x* without the protection of a lock on ^*y*, the rollback of ^*x* does not work.

On the rollback of **Set** and **Kill** operations, if the current value of the data item is what the operation set it to, the value is reset to what it was before the operation. If the current value is different from what the specific **Set** or **Kill** operation set it to, the current value is left unchanged.

If a data item is sometimes modified inside a transaction, and sometimes modified outside of a transaction and outside the protection of a **Lock** command, rollback is not guaranteed to work. To be effective, locks must be used everywhere a data item is modified.

2.5.2.5 Journal Discontinuity Breaks Rollback

Rollback depends on the reliability and completeness of the journal. If something interrupts the continuity of the journal data, rollbacks do not succeed past the discontinuity. InterSystems IRIS silently ignores this type of transaction rollback.

A journal discontinuity can be caused by executing **^JRNSTOP** while InterSystems IRIS is running, by deleting the Write Image Journal (WIJ) file after an InterSystems IRIS shutdown and before restart, or by an I/O error during journaling on a system that is not set to freeze the system on journal errors.

2.5.2.6 ECP Can Miss Error After Recovery

A **Set** or **Kill** operation completes on a data server, but receives an error. The data server crashes after completing that packet, but before delivering that packet to the application server system.

ECP recovery does not replay this packet, but the application server has not found out about the error; resulting in the application server missing **Set** or **Kill** operations on the data server.

2.5.2.7 Partial Set or Kill Leads to Journal Mismatch

There are certain cases where a **Set** or **Kill** operation can be journaled successfully, but receive an error before actually modifying the database. Given the particular way rollback of a data item is defined, this should not ever break transaction rollback; but the state of a database after a journal restore may not match the state of that database before the restore.

2.5.2.8 Loose Ordering in Cluster Failover or Restore

Cluster dejournaling is loosely ordered. The journal files from the separate cluster members are only synchronized wherever a lock, a **$Increment**, or a journal marker event occurs. This affects the database state after either a cluster failover or a cluster crash where the entire cluster must be brought down and restored. The database may be restored to a state that is different from the state just before the crash. The $Increment Ordering Guarantee places additional constraints on how different the restored database can be from its original form before the crash.

Process B's ability to view the data modified by *Process A* does not ensure that **Set** operations from *Process B* are restored after the **Set** operations from *Process A*. Only a **Lock** or a **$Increment** operation can ensure proper ordering of competing **Set** and **Kill** operations from two different processes during cluster failover or cluster recovery.

2.5.2.9 Dirty Data Reads When Cluster Member Crashes

A cluster *Member A* completes updates in *Transaction T1*, and that system commits that transaction, but in non-synchronous transaction commit mode. *Transaction T2* on a different cluster *Member B* acquires the locks once owned by *Transaction T1*. Cluster *Member A* crashes before all the information from *Transaction T1* is written to disk.

Transaction T1 is rolled back as part of cluster failover. However, *Transaction T2* on *Member B* could have seen data from *Transaction T1* that later was rolled back as part of cluster failover, despite following the rules of the locking protocol. Additionally, if *Transaction T2* has modified some of the same data items as *Transaction T1*, the rollback of *Transaction T1* may fail because only some of the transaction data has rolled back.

A workaround is to use synchronous commit mode for transactions on cluster *Member A*. When using synchronous commit mode, *Transaction T1* is durable on disk before its locks are released, so *Transaction T1* is not rolled back once the application sees that it is complete.

2.5.2.10 Dirty Data Reads in ECP Without Locking

If an incoming ECP transaction reads data without locking, it may see data that is not durable on disk which may disappear if the data server crashes. It can only see such data when the data location is set by other ECP connections or by the local data server system itself. It can never see nondurable data that is set by this connection itself. There is no possibility of seeing nondurable data when locking is used both in the process reading the data and the process writing the data. This is a violation of the In-order Updates Guarantee and there is no easy workaround other than to use locking.

2.5.2.11 Asynchronous TCommit Converts to Rollback

If the data server side of a transaction receives an asynchronous error condition, such as a `<FILEFULL>`, while updating a database, and the application server does not see that error until the **TCommit**, the transaction is automatically rolled back on the data server. However, rollbacks are synchronous while **TCommit** operations are usually asynchronous because the rollback will be changing blocks the application server should be notified of before the application server process surrenders any locks.

The data server and the database are fine, but on the application server if the locks get traded to another process you may see temporarily see data that is about to be rolled back. However, the application server does not usually do anything that causes asynchronous errors.

3

Horizontally Scaling for Data Volume with Sharding

This page describes the deployment and use of an InterSystems IRIS sharded cluster.

3.1 Overview of InterSystems IRIS Sharding

Sharding is a significant horizontal scalability feature of InterSystems IRIS. An InterSystems IRIS sharded cluster partitions both data storage and caching across a number of servers, providing flexible, inexpensive performance scaling for queries and data ingestion while maximizing infrastructure value through highly efficient resource utilization. Sharding is easily combined with the considerable vertical scaling capabilities of InterSystems IRIS, greatly widening the range of workloads for which InterSystems IRIS can provide solutions.

- Elements of Sharding
- Evaluating the Benefits of Sharding
- Namespace-level Sharding Architecture

Note: For a brief introduction to sharding that includes a hands-on exploration of deploying and using a sharded cluster, see Demo: Scaling for Data Volume with an InterSystems IRIS Sharded Cluster.

3.1.1 Elements of Sharding

Horizontally scaling via sharding can benefit a wide range of applications, but provides the greatest gains in use cases involving one or both of the following:

- Large amounts of data retrieved from disk, complex processing of data, or both, for example as in analytic workloads
- High-volume, high-velocity data ingestion

Sharding horizontally partitions large database tables and their associated indexes across multiple InterSystems IRIS instances, called *data nodes*, while allowing applications to access these tables through any one of those instances. Together, the data nodes form a *sharded cluster*. This architecture provides the following advantages:

- Queries against a *sharded table* are run in parallel on all of the data nodes, with the results merged, aggregated, and returned as full query results to the application.

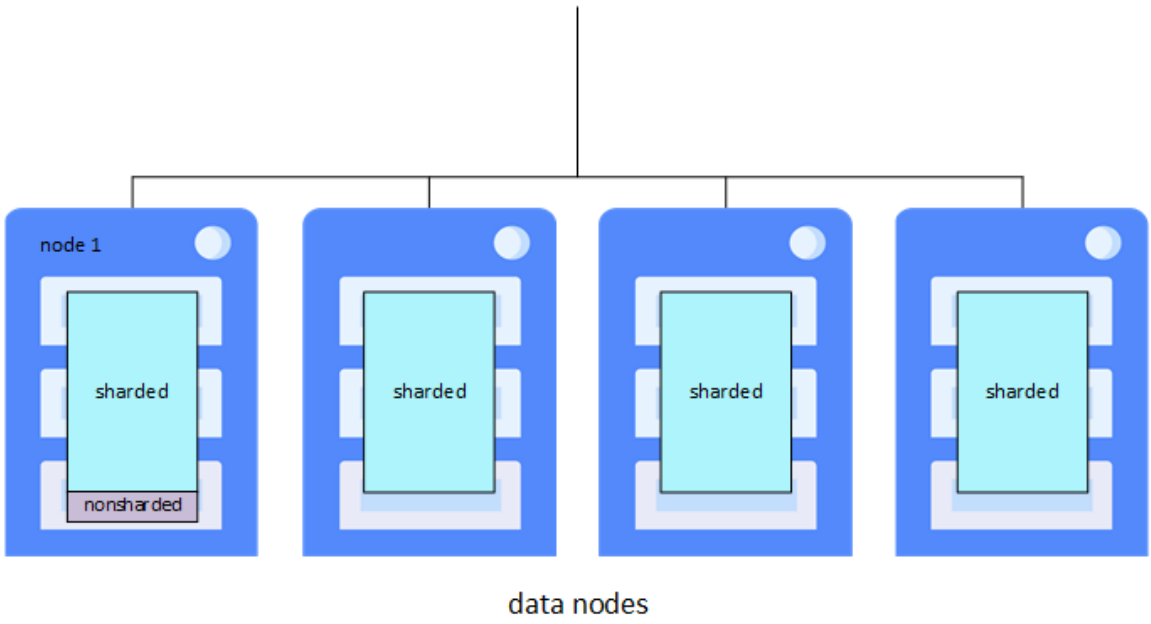
- Because the data partitions are hosted by separate instances, each has a dedicated cache to serve its own partition of the data set, rather than a single instance's cache serving the entire data set.

With sharding, the performance of queries against large tables is no longer constrained by the resources of a single system. By distributing both query processing and caching across multiple systems, sharding provides near-linear scaling of both compute and memory resources, allowing you to design and maintain a cluster tailored to your workload. When you scale out by adding data nodes, sharded data can be rebalanced across the cluster. The distributed data layout can be further exploited for parallel data loading and with third party frameworks.

A *shard* is a subset of a table's rows, with each row contained within exactly one shard, and all shards of a table containing roughly the same number of rows. Each data node hosts a *data shard*, which is comprised of one shard of each sharded table on the cluster. A federated software component called the *sharding manager* keeps track of which shards (and therefore which table rows) are located on which data nodes and directs queries accordingly, as well as managing other sharded operations. Each table is automatically horizontally partitioned across the data nodes by using one of its fields as a *shard key*, which provides a deterministic method of distributing data evenly. A shard key is typically the table's RowID (the default), but can also be a user-defined field or set of fields.

While sharded data is physically partitioned across the data nodes, it is all logically visible from on any data node (as are nonsharded data, metadata, and code). Each data node has a *cluster namespace* (identically named across the cluster) that provides transparent access to all data and code on the cluster; applications can connect to any node's cluster namespace and experience the full dataset as if it were local. Application connections should therefore be load balanced across all of the data nodes in the cluster to take greatest advantage of parallel query processing and partitioned caching.

Figure 3–1: Basic sharded cluster



Nonsharded data is stored only on the first data node configured (called *data node 1*, or just node 1). This distinction is transparent to the user except for the fact that more data is stored on node 1, but this difference is typically small. From the perspective of the application SQL, the distinction between sharded and nonsharded tables is transparent.

InterSystems IRIS mirroring can be used to provide high availability for the data nodes in a sharded cluster; a mirrored failover pair of InterSystems IRIS instances can be added to a cluster as easily as a single instance. For more information on deploying a mirrored sharded cluster, see Mirror Data Nodes for High Availability.

For advanced use cases in which extremely low query latencies are required, potentially at odds with a constant influx of data, *compute nodes* can be added to provide a transparent caching layer for servicing queries, separating the query and data ingestion workloads and improving the performance of both. Assigning multiple compute nodes per data node can

further improve the cluster's query throughput. For more information about compute nodes and instructions for deploying them, see Deploy Compute Nodes for Workload Separation and Increased Query Throughput.

3.1.2 Evaluating the Benefits of Sharding

InterSystems IRIS sharding can benefit a wide range of applications, but provides the greatest gains in use cases involving the following:

- Relatively large data sets, queries that return large amounts of data, or both.

  Sharding scales caching capacity to match data size by partitioning the cache along with the data, leveraging the memory resources of multiple systems. Each data node dedicates its database cache (global buffer pool) to a fraction of the data set, as compared to a single instance's database cache being available for all of the data. The resulting improvement becomes most evident when the data in regular use is too big to fit in the database cache of a single nonsharded instance.

- A high volume of complex queries doing large amounts of data processing.

  Sharding scales query processing throughput by decomposing queries and executing them in parallel across multiple data nodes, leveraging the computing resources of multiple systems. The resulting improvement is most evident when queries against the cluster:

  – Read large amounts of data from persistent storage, and in particular have a high ratio of data retrieved to results returned.

  – Involve significant compute work (including aggregation, grouping, and sorting)

- High-volume or high-speed data ingestion, or a combination.

  Sharding scales data ingestion through the InterSystems IRIS JDBC driver's use of direct connections to the data nodes for parallel loading, distributing ingestion across multiple instances. If the data can be assumed to be validated and uniqueness checking omitted, gains are enhanced.

Each of these factors on its own influences the potential gain from sharding, but the benefit may be enhanced where they combine. For example, a combination of large amounts of data ingested quickly, large data sets, and complex queries that retrieve and process a lot of data makes many of today's analytic workloads very good candidates for sharding.

As previously noted, and discussed in more detail in Planning an InterSystems IRIS Sharded Cluster, combining InterSystems IRIS sharding with the use of vertical scaling to address some of the factors described in the foregoing may be most beneficial under many circumstances.

Note: In the current release, sharding does not support workloads involving complex transactions requiring atomicity, and a sharded cluster cannot be used for such workloads.

3.1.3 Namespace-level Sharding Architecture

Older versions of this document described a sharding architecture involving a larger and different set of node types (shard master data server, shard data server, shard query server). This *namespace-level architecture* remains in place as the transparent foundation of the new *node-level* architecture, and is fully compatible with it. In the node-level architecture, the cluster namespace (identically named across the cluster) provides transparent access to all sharded and nonsharded data and code on the cluster; the master namespace, now located on the first data node, still provides access to metadata, nonsharded data, and code, but is fully available to all data nodes. This arrangement provides a more uniform and straightforward model that is simpler and more convenient to deploy and use.

You can deploy a namespace-level sharded cluster using the Sharding Configuration page of the Management Portal and the %SYSTEM.Sharding; these procedures are described in Deploying the Namespace-level Architecture.

3.2 Deploying the Sharded Cluster

To provide you with simple ways to create your first cluster for evaluation and testing purposes, this section includes manual procedures for deploying a basic InterSystems IRIS sharded cluster consisting of nonmirrored data nodes, using either the Management Portal or the %SYSTEM.Cluster to configure the cluster. These procedures are extended for adding compute nodes and mirroring in subsequent sections, as follows:

- Compute nodes can be easily added to an existing cluster; if you are considering deploying compute nodes in production, the best approach is typically to evaluate the operation of your data node-only cluster before deciding whether it can benefit from their addition. For more information on planning and adding compute nodes, see Plan Compute Nodes and Deploy Compute Nodes for Workload Separation and Increased Query Throughput.
- Other than adding failover (and optionally disaster recovery) capability to each data node and the possible minor impact of latency between failover members, a mirrored sharded cluster performs in exactly the same way as a nonmirrored cluster of the same number of data nodes. If you are interested in deploying a mirrored sharded cluster, see Mirror Data Nodes for High Availability for procedures.

InterSystems IRIS data platform also offers several methods for automated deployment of sharded clusters, which greatly simplify the process of deploying clusters of varying topology on a variety of platforms, including on-premises hardware, public and private clouds, and Kubernetes.

Regardless of the method you use to deploy a sharded cluster, the first steps are to decide how many data nodes are to be included in the cluster and plan the sizes of the database caches and globals databases on those nodes. When deploying manually, you also need to identify or provision the infrastructure that will host the cluster and deploy InterSystems IRIS on the hosts before configuring the cluster. Therefore, to deploy your basic cluster using the manual procedures in this section, start with these steps and then choose a manual configuration method, as follows:

1. Plan data nodes
2. Estimate the database cache and database sizes
3. Provision or identify the infrastructure
4. Deploy InterSystems IRIS on the data node hosts
5. Configure the cluster using either:
   - the Management Portal
   - the %SYSTEM.Cluster API

Important: The instructions in this section:

- Do not describe the deployment of a mirrored sharded cluster or of compute nodes. Mirrored deployment is covered in Mirror for High Availability, while Deploy Compute Nodes for Workload Separation and Increased Query Throughput provides instructions for using the API to add compute nodes to a basic cluster.
- Deploy a node-level sharded cluster; for instructions for deploying a namespace-level cluster, see Deploying the Namespace-level Architecture.

Note: These instructions assume you will deploy new InterSystems IRIS instances on the hosts you provisioned or identified before configuring the cluster, but can be adapted for use with existing instances, as long as the instances to be configured as cluster nodes and their hosts adhere to the requirements and guidelines described in the first four steps.

For an important discussion of load balancing a web server tier distributing application connections across data and (where applicable) compute nodes, see Load Balancing, Failover, and Mirrored Configurations.

For an important discussion of performance planning, including memory management and scaling, CPU sizing and scaling, and other considerations, see System Resource Planning and Management.

In the most typical sharded cluster configuration, each cluster node consists of one InterSystems IRIS instance on one physical or virtual system. This configuration is assumed in the procedures provided on this page.

InterSystems recommends the use of an LDAP server to implement centralized security across the nodes of a sharded cluster. For information about using LDAP with InterSystems IRIS, see LDAP Guide.

HealthShare Health Connect does not support sharding.

3.2.1 Automated Deployment Methods for Clusters

In addition to the manual procedures outlined in this section, InterSystems IRIS Data platform provides two methods for automated deployment of sharded clusters that are fully operational following deployment.

3.2.1.1 Deploy a Sharded Cluster Using the InterSystems Kubernetes Operator (IKO)

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 isn't required to deploy InterSystems IRIS under Kubernetes, but it greatly simplifies the process and adds InterSystems IRIS-specific cluster management capabilities to Kubernetes, enabling 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.

3.2.1.2 Deploy a Sharded Cluster Using 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 when restarting 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.

Using configuration merge, you can deploy a sharded cluster by calling separate merge files for the different node types, sequentially deploying data node 1, then the remaining data nodes, then (optionally) the compute nodes. (Because the instance configured as data node 1 must be running before the other data nodes can be configured, you must ensure that this instance is deployed and successfully started before other instances are deployed.) You can also automatically deploy a cluster of data nodes (optionally mirrored) using a single merge file if the deployment hosts have names ending in the regular expression you specify; following deployment of the data nodes, you can optionally deploy compute nodes using a separate merge file.

The IKO, described above, incorporates the configuration merge feature.

For information about using configuration merge in general and to deploy sharded clusters in particular, see Automating Configuration of InterSystems IRIS with Configuration Merge.

3.2.2 Plan Data Nodes

Depending on the anticipated working set of the sharded data you intend to store on the cluster and the nature of the queries you will run against it, as few as four data nodes may be appropriate for your cluster. You can always add data nodes to an existing cluster and rebalance the sharded data (see Add Data Nodes and Rebalance Data), so erring on the conservative side is reasonable.

A good basic method for an initial estimate of the ideal number of data nodes needed for a production configuration (subject to resource limitations) is to calculate the total amount of database cache needed for the cluster and then determine which combination of server count and memory per server is optimal in achieving that, given your circumstances and resource availability. This is not unlike the usual sizing process, except that it involves dividing the resources required across multiple systems. (For an important discussion of performance planning, including memory management and scaling, CPU sizing and scaling, and other considerations, see System Resource Planning and Management.)

The size of the database cache required starts with your estimation of the total amount of sharded data you anticipate storing on the cluster, and of the amount of nonsharded data on the cluster that will be frequently joined with sharded data. You can then use these totals to estimate the working sets for both sharded data and frequently joined nonsharded data, which added together represent the total database caching capacity needed for all the data nodes in the cluster. This calculation is detailed in Planning an InterSystems IRIS Sharded Cluster.

Considering all your options regarding both number of nodes and memory per node, you can then configure enough data nodes so that the database cache (global buffer pool) on each data node equals, or comes close to equalling, its share of that capacity. Under many scenarios, you will be able to roughly determine the number of data nodes to start with simply by dividing the total cache size required by the memory capacity of the systems you available to deploy as cluster nodes.

All data nodes in a sharded cluster should have identical or at least closely comparable specifications and resources; parallel query processing is only as fast as the slowest data node. In addition, the configuration of all InterSystems IRIS instances in the cluster should be consistent; database settings such as collation and those SQL settings configured at instance level (default date format, for example) should be the same on all nodes to ensure correct SQL query results. Standardized procedures and the available automated deployment methods for sharded clusters can help ensure this consistency.

Because applications can connect to any data node's cluster namespace and experience the full dataset as if it were local, the general recommended best practice is to load balance application connections across all of the data nodes in a cluster. The IKO can automatically provision and configure a load balancer for the data nodes as needed under typical scenarios; if deploying a sharded cluster by other means, a load balancing mechanism is required. For an important discussion of load balancing a web server tier distributing application connections across data nodes, see Load Balancing, Failover, and Mirrored Configurations.

3.2.3 Estimate the Database Cache and Database Sizes

Before deploying your sharded cluster, determine the size of the database cache to be allocated on each data node. It is also useful to know the expected size of the data volume needed for the default globals database on each data node, so you can ensure that there is enough free space for expected growth.

When you deploy a sharded cluster using an automated deployment method, you can specify these settings as part of deployment. When you deploy manually using the Sharding API or the Management Portal, you can specify database cache size of each instance before configuring the sharded cluster, and specify database settings in your calls. Both manual deployment methods provide default settings.

Bear in mind that the sizes below are guidelines, not requirements, and that your estimates for these numbers are likely to be adjusted in practice.

3.2.3.1 Database Cache Sizes

As described in Planning an InterSystems IRIS Sharded Cluster, the amount of memory that should ideally be allocated to the database cache on a data node is that node's share of the total of the expected sharded data working set, plus the overall expected working set of nonsharded data frequently joined to sharded data.

3.2.3.2 Globals Database Sizes

As described in Planning an InterSystems IRIS Sharded Cluster, the target sizes of the default globals databases are as follows:

- For the cluster namespace — Each server's share of the total size of the sharded data, according to the calculation described in that section, plus a margin for greater than expected growth.
- For the master namespace on node 1 — The total size of nonsharded data, plus a margin for greater than expected growth.

All deployment methods configure the sizes of these databases by default, so there is no need for you to do so. However, you should ensure that the storage on which these databases are located can accommodate their target sizes.

3.2.4 Provision or Identify the Infrastructure

Provision or identify the needed number of networked host systems (physical, virtual, or cloud) — one host for each data node.

Important: All data nodes in a sharded cluster should have identical or at least closely comparable specifications and resources; parallel query processing is only as fast as the slowest data node. (The same is true of compute nodes, although storage is not a consideration in their case.)

The recommended best practice is to load balance application connections across all of the data nodes in a cluster.

To maximize the performance of the cluster, it is a best practice to maximize network throughput by configuring low-latency network connections between all of the data nodes, for example by locating them on the same subnet in the same data center or availability zone. This procedure assumes that the data nodes are mutually accessible through TCP/IP, with a recommended minimum network bandwidth of 1 GB between all nodes and preferred bandwidth of 10 GB or more, if available.

3.2.5 Deploy InterSystems IRIS on the Data Node Hosts

This procedure assumes that each system hosts or will host a single InterSystems IRIS instance. You can configure existing instances instead of newly deployed instances in the final step, as long as each instance and its host meet the requirements described in this step.

Important: All InterSystems IRIS instances in a sharded cluster must be of the same version, and all must have sharding licenses.

All instances should have their database directories and journal directories located on separate storage devices, if possible. This is particularly important when high volume data ingestion is concurrent with running queries. For guidelines for file system and storage configuration, including journal storage, see Storage Planning, File System Separation, and Journaling Best Practices.

The configuration of all InterSystems IRIS instances in the cluster should be consistent; database settings such as collation and those SQL settings configured at the instance level (default date format, for example) should be the same on all nodes to ensure correct SQL query results.

On each host system, do the following:

1. Deploy an instance of InterSystems IRIS, either by creating a container from an InterSystems-provided image (as described in Running InterSystems Products in Containers) or by installing InterSystems IRIS from a kit (as described in the Installation Guide).
2. Ensure that the storage device hosting the instance's databases is large enough to accommodate the target globals database size, as described in Estimate the Database Cache and Database Sizes.
3. Allocate the database cache (global buffer pool) for the instance according to the size you determined in Estimate the Database Cache and Database Sizes. For the Management Portal procedure for allocating the database cache, see Memory and Startup Settings; you can also allocate the cache using the globals parameter, either by editing the instance's configuration parameter file (CPF) or, on UNIX® and Linux platforms, deploying the instance with the desired value using a configuration merge file.

   In some cases, it may also be advisable to increase the size of the shared memory heap on the cluster members. The shared memory heap can be configured either using the Management Portal, as described for the gmheap parameter, or in the instance's CPF file using gmheap, as described above for globals.

   Note: For general guidelines for estimating the memory required for an InterSystems IRIS instance's routine and database caches as well as the shared memory heap, see Shared Memory Allocations.

Finally, configure the deployed instances as a sharded cluster using either the Management Portal (the next section) or the %SYSTEM.Cluster API (the one after that).

3.2.6 Configure the Cluster Using the Management Portal

Once you have completed the first four steps (planning the data nodes, estimating the database cache and database sizes, provisioning or identifying the infrastructure, and deploying InterSystems IRIS on the data node hosts), use this procedure to configure the instances you deployed (or the existing instances you prepared) in the previous step as a basic InterSystems IRIS sharded cluster of data nodes using the Management Portal.

Configure each node in the cluster using the following steps.

1. Configure node 1
2. Configure the remaining data nodes

For information about opening the Management Portal in your browser, see the instructions for an instance deployed in a container or one installed from a kit.

Following the procedure, other Management Portal sharded cluster options are briefly discussed.

3.2.6.1 Configure Data Node 1

A sharded cluster is initialized when you configure the first data node, which is referred to as *data node 1*, or simply *node 1*. This data node differs from the others in that it stores the cluster's nonsharded data, metadata, and code, and hosts the master namespace that provides all of the data nodes with access to that data. This distinction is completely transparent to the user except for the fact that more data is stored on the first data node, a difference that is typically small.

Once you have completed the first four steps (planning the data nodes, estimating the database cache and database sizes, provisioning or identifying the infrastructure, and deploying InterSystems IRIS on the data node hosts), use this procedure to configure the instances you deployed (or the existing instances you prepared) in the previous step as a basic InterSystems IRIS sharded cluster of data nodes using the Management Portal.

To configure node 1, follow these steps:

1. Open the Management Portal for the instance, select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK**. (The value of the **Maximum Number of ECP Connections** setting need not be changed as the default is appropriate for virtually all clusters.)
2. Restart the instance. (There is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted.)
3. Navigate to the Configure Node-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Node-Level**) and click the **Configure** button.
4. On the CONFIGURE NODE-LEVEL CLUSTER dialog, select **Initialize a new sharded cluster on this instance** and respond to the prompts that display as follows:
   - Select a **Cluster namespace** and **Master namespace** from the respective drop-downs, which both include
     - The default names (**IRISCLUSTER** and **IRISDM**) of new namespaces to be created, along with their default globals databases.
     - All eligible existing namespaces.

     Initializing a sharded cluster creates by default cluster and master namespaces named **IRISCLUSTER** and **IRISDM**, respectively, as well as their default globals databases and the needed mappings. However, to control the names of the cluster and master namespaces and the characteristics of their globals databases, you can create one or both namespaces and their default databases before configuring the cluster, then specify them during the procedure. For example, given the considerations discussed in Globals Database Sizes, you may want to do this to control the characteristics of the default globals database of the cluster namespace, or *shard database*, which is replicated on all data nodes in the cluster.

     Note: If the default globals database of an existing namespace you specify as the cluster namespace contains any globals or routines, initialization will fail with an error.

   - In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, enter the IP address at the **Override hostname** prompt.
   - Do not select **Enable Mirroring**; the procedure for deploying a mirrored sharded cluster is provided in Mirror for High Availability
5. Click **OK** to return to the Configure Node-Level page, which now includes two tabs, **Shards** and **Sharded Tables**. Node 1 is listed under **Shards**, including its cluster address (which you will need in the next procedure), so you may want to leave the Management Portal for node 1 open on the Configure Node-Level page, for reference. Click **Verify Shards** to verify that node 1 is correctly configured.

3.2.6.2 Configure the Remaining Data Nodes

Once node 1 has been configured, configure each additional data node using these steps:

1. Open the Management Portal for the instance, select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK**. (The value of the **Maximum Number of ECP Connections** setting need not be changed as the default is appropriate for virtually all clusters.)
2. Restart the instance. (There is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted.)
3. Navigate to the Configure Node-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Node-Level**) and click the **Configure** button.

4. On the CONFIGURE NODE-LEVEL CLUSTER dialog, select **Add this instance to an existing sharded cluster** and respond to the prompts that display as follows:

   - Enter the cluster URL, which is the address displayed for node 1 on the **Shards** tab of the Configure Node-Level page (as described in the previous procedure). The cluster URL consists of the node 1 host's identifier — either hostname or IP address (if you provided the IP address in the previous procedure) — plus the InterSystems IRIS instance's superserver port and the name of the cluster namespace, for example clusternode011.acmeinternal.com:1972:IRISCLUSTER. (The name of the cluster namespace can be omitted if it is the default **IRISCLUSTER**.)

     Note: From the perspective of another node (which is what you need in this procedure), the superserver port of a containerized InterSystems IRIS instance depends on which host port the superserver port was published or exposed as when the container was created. For details on and examples of this, see Running an InterSystems IRIS Container with Durable %SYS and Running an InterSystems IRIS Container: Docker Compose Example and see Container networking in the Docker documentation.

     The default superserver port number of a kit-installed InterSystems IRIS instance that is the only such on its host is 1972; when multiple instances are installed, superserver port numbers range from 51776 upwards. To see or change an the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal.

   - Select **data** at the **Role** prompt to configure the instance as a data node.
   - In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, enter the IP address at the **Override hostname** prompt.
   - Do not select **Mirrored cluster**; the procedure for deploying a mirrored sharded cluster is provided in Mirror for High Availability

5. Click **OK** to return to the Configure Node-Level page, which now includes two tabs, **Shards** and **Sharded Tables**. The data nodes you have configured so far are listed under **Shards**, starting with node 1. Click **Verify Shards** to verify that the data node is correctly configured and can communicate with the others.

   Note: If you have many data nodes to configure, you can make the verification operation automatic by clicking the **Advanced Settings** button and selecting **Automatically verify shards on assignment** on the ADVANCED SETTINGS dialog. Other settings in this dialog should be left at the defaults when you deploy a sharded cluster.

3.2.6.3 Management Portal Sharding Options

Use the **Rebalance** button to display the REBALANCE dialog, which lets you redistribute sharded data evenly across all data nodes, including those recently added to an existing cluster. (You cannot query sharded tables while a rebalancing operation is running.) For details about rebalancing data, see Add Data Nodes and Rebalance Data.

The ADVANCED SETTINGS dialog, which you can display by clicking the **Advanced Settings** button on the Configure Node-Level page, provides the following options:

- Select **Automatically verify shards on assignment** to make verification automatic following configuration of a new cluster node.
- Change the cluster's **sharded query execution mode** from the default asynchronous to synchronous; for a discussion of the effects of this, see Including Compute Nodes in Mirrored Clusters for Transparent Query Execution Across Failover.
- Change the **shard connection timeout** from the default of 60 seconds.

- Use **data node 1 host's IP address** to identify it in cluster communications, rather than its hostname.
- Change the number of times the cluster should **retry connecting to a mirrored shard** from the default of one.
- Change the maximum number of ECP connections for the local node. The total number of nodes in the cluster cannot exceed this setting, which is set to 64 when you use the **Enable Sharding** Management Portal option, as described at the start of each of the two steps in this section.
- Set the cluster to **ignore errors during DROP TABLE operations**.

The **Sharded Tables** tab on the Configure Node-Level page display information about all of the sharded tables on the cluster.

3.2.7 Configure the Cluster Using the %SYSTEM.Cluster API

Use the following procedure to configure the instances you deployed (or the existing instances you prepared) in the previous step as a basic InterSystems IRIS sharded cluster of data nodes using the %SYSTEM.Cluster API. You will probably find it useful to refer to the %SYSTEM.Cluster class documentation in the InterSystems Class Reference. (As with all classes in the %SYSTEM package, the %SYSTEM.Cluster methods are also available through $SYSTEM.Cluster.)

Configure each node in the cluster using the following steps:

1. Configure node 1
2. Configure the remaining data nodes

3.2.7.1 Configure Node 1

A sharded cluster is initialized when you configure the first data node, which is referred to as *data node 1*, or simply *node 1*. This data node differs from the others in that it stores the cluster's nonsharded data, metadata, and code, and hosts the master namespace that provides all of the data nodes with access to that data. This distinction is completely transparent to the user except for the fact that more data is stored on the first data node, a difference that is typically small.

To configure node 1, open the InterSystems Terminal for the instance and call the **$SYSTEM.Cluster.Initialize()** method, for example:

```
set status = $SYSTEM.Cluster.Initialize()
```

Note: To see the return value (for example, 1 for success) for the each API call detailed in these instructions, enter:

```
zw status
```

Reviewing **status** after each call is a good general practice, as a call might fail silently under some circumstances. If a call does not succeed (**status** is not **1**), display the user-friendly error message by entering:

```
do $SYSTEM.Status.DisplayError(status)
```

The **Initialize()** call creates the master and cluster namespaces (**IRISDM** and **IRISCLUSTER**, respectively) and their default globals databases, and adds the needed mappings. Node 1 serves as a template for the rest of the cluster; the name of the cluster namespace, the characteristics of its default globals database (also called the *shard database*), and its mappings are directly replicated on the second data node you configure, and then directly or indirectly on all other data nodes. The SQL configuration settings of the instance are replicated as well.

To control the names of the cluster and master namespaces and the characteristics of their globals databases, you can specify existing namespaces as the cluster namespace, master namespace, or both by including one or both names as arguments. For example:

```
set status = $SYSTEM.Cluster.Initialize("CLUSTER","MASTER",,)
```

When you do this, the existing default globals database of each namespace you specify remains in place. This allows you to control the characteristics of the shard database, which are then replicated on other data nodes in the cluster.

Note: If the default globals database of an existing namespace you specify as the cluster namespace contains any globals or routines, initialization will fail with an error.

By default, any host can become a cluster node; the third argument to **Initialize()** lets you specify which hosts can join the cluster by providing a comma-separated list of IP addresses or hostnames. Any node not in the list cannot join the cluster.

In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, include the IP address as the fourth argument. (You cannot supply a hostname as this argument, only an IP address.) In either case, you will use the host identifier (hostname or IP address) to identify node 1 when configuring the second data node; you will also need the superserver (TCP) port of the instance.

Note: From the perspective of another node (which is what you need in this procedure), the superserver port of a containerized InterSystems IRIS instance depends on which host port the superserver port was published or exposed as when the container was created. For details on and examples of this, see Running an InterSystems IRIS Container with Durable %SYS and Running an InterSystems IRIS Container: Docker Compose Example and see Container networking in the Docker documentation.

The default superserver port number of a kit-installed InterSystems IRIS instance that is the only such on its host is 1972. To see or set the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal. (For information about opening the Management Portal in your browser, see the instructions for an instance deployed in a container or one installed from a kit.)

The **Initialize()** method returns an error if the InterSystems IRIS instance is already a node in a sharded cluster, or is a mirror member.

3.2.7.2 Configure the Remaining Data Nodes

To configure each additional data node, open the Terminal for the InterSystems IRIS instance and call the **$SYSTEM.Cluster.AttachAsDataNode()** method, specifying the hostname of an existing cluster node (node 1, if you are configuring the second node) and the superserver port of its InterSystems IRIS instance, for example:

```
set status = $SYSTEM.Cluster.AttachAsDataNode("IRIS://datanode1:1972")
```

If you supplied an IP address as the fourth argument to **Initialize()** when initializing node 1, use the IP address instead of the hostname to identify node 1 in the first argument, for example:

```
set status = $SYSTEM.Cluster.AttachAsDataNode("IRIS://100.00.0.01:1972")
```

Note: For important information about determining the correct superserver port to specify, see the previous step, Configure Node 1.

The **AttachAsDataNode()** call does the following:

- Creates the cluster namespace and shard database, configuring them to match the settings on the template node (specified in the first argument), as described in Configure Node 1, and creating the needed mappings, including those to the globals and routines databases of the master namespace on node 1 (including any user-defined mappings).
- Sets all SQL configuration options to match the template node.
- Because this node may later be used as the template node for **AttachAsDataNode()**, sets the list of hosts eligible to join the cluster to those you specified (if any) in the **Initialize()** call on node 1.

Note: If a namespace of the same name as the cluster namespace on the template node exists on the new data node, it and its globals database are used as the cluster namespace and shard database, and only the mappings are replicated. If the new node is subsequently used as the template node, the characteristics of these existing elements are replicated.

The **AttachAsDataNode()** call returns an error if the InterSystems IRIS instance is already a node in a sharded cluster or is a mirror member, or if the template node specified in the first argument is a mirror member.

As noted in the previous step, the hostname known to InterSystems IRIS may not resolve to an appropriate address, or no hostname is available. To have other cluster nodes communicate with this node using its IP address instead, include the IP address as the second argument. (You cannot supply a hostname as this argument, only an IP address.)

When you have configured all of the data nodes, you can call the **$SYSTEM.Cluster.ListNodes()** method to list them, for example:

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    Host          Port
1       Data        datanode1     1972
2       Data        datanode2     1972
3       Data        datanode3     1972
```

As shown, data nodes are assigned numeric IDs representing the order in which they are attached to the cluster.

The recommended best practice is to load balance application connections across all of the data nodes in a cluster.

For information about adding compute nodes to your cluster, see Deploy Compute Nodes for Workload Separation and Increased Query Throughput.

3.3 Creating Sharded Tables and Loading Data

Once the cluster is fully configured, you can plan and create the sharded tables and load data into them. The steps involved are as follows:

- Evaluate existing tables for sharding
- Create sharded tables
- Load data onto the cluster
- Create and load nonsharded tables

3.3.1 Evaluate Existing Tables for Sharding

Although the ratio of sharded to nonsharded data on a cluster is typically high, when planning the migration of an existing schema to a sharded cluster it is worth remembering that not every table is a good candidate for sharding. In deciding which of your application's tables to define as sharded tables and which to define as nonsharded tables, your primary considerations should be improving query performance and/or the rate of data ingestion, based on the following factors (which are also discussed in Evaluating the Benefits of Sharding). As you plan, remember that the distinction between sharded and nonsharded tables is totally transparent to the application SQL; like index selection, sharding decisions have implications for performance only.

- Overall size — All other things being equal, the larger the table, the greater the potential gain.
- Data ingestion — Does the table receive frequent and/or large INSERT statements? Parallel data loading means sharding can improve their performance.

- Query volume — Which tables are queried most frequently on an ongoing basis? Again, all other things being equal, the higher the query volume, the greater the potential performance improvement.
- Query type — Among the larger tables with higher query volume, those that frequently receive queries that read a lot of data (especially with a high ratio of data read to results returned) or do a lot of computing work are excellent candidates for sharding. For example, is the table frequently scanned by broad SELECT statements? Does it receive many queries involving aggregate functions?

Having identified some good candidates for sharding, review the following considerations:

- Frequent joins — As discussed in Choose a Shard Key, tables that are frequently joined can be sharded with equivalent shard keys to enable cosharded joins, so that joining can be performed locally on individual shards, enhancing performance. Review each frequently-used query that joins two large tables with an equality condition to evaluate whether it represents an opportunity for a cosharded join. If the queries that would benefit from cosharding the tables represent a sizeable portion of your overall query workload, these joined tables are good candidates for sharding.

  However, when a large table is frequently joined to a much smaller one, sharding the large one and making the small one nonsharded may be most effective. Careful analysis of the frequency and query context of particular joins can be very helpful in choosing which tables to shard.
- Unique constraints — A unique constraint on a sharded table can have a significant negative impact on insert/update performance unless the shard key is a subset of the unique key; see Choose a Shard Key for more information.

Important: Regardless of other factors, tables that are involved in complex transactions requiring atomicity should never be sharded.

3.3.2 Create Sharded Tables

Sharded tables (as well as nonsharded tables) can be created in the cluster namespace on any node, using a SQL CREATE TABLE statement containing a sharding specification, which indicates that the table is to be sharded and with what shard key — the field or fields that determine which rows of a sharded table are stored on which shards. Once the table is created with the appropriate shard key, which provides a deterministic method of evenly distributing the table's rows across the shards, you can load data into it using INSERT and dedicated tools.

3.3.2.1 Choose a Shard Key

By default, when you create a sharded table and do not specify a shard key, data is loaded into it using the system-assigned RowID as the shard key; for example, with two shards, the row with RowID=1 would go on one shard and the one with RowID=2 would go on the other, and so on. This is called a *system-assigned shard key*, or *SASK*, and is often the simplest and most effective approach because it offers the best guarantee of an even distribution of data and allows the most efficient parallel data loading.

Note: By default, the RowID field is named ID and is assigned to column 1. If a user-defined field named ID is added, the RowID field is renamed to ID1 when the table is compiled, and it is the user-defined ID field that is used by default when you shard without specifying a key.

You also have the option of specifying one or more fields as the shard key when you create a sharded table; this is called a *user-defined shard key*, or *UDSK*. You might have good opportunities to use UDSKs if your schema includes semantically meaningful unique identifiers that do not correspond to the RowID, for example when several tables in a schema contain an accountnumber field.

An additional consideration concerns queries that join large tables. Every sharded query is decomposed into shard-local queries, each of which is run independently and locally on its shard and needs to see only the data that resides on that shard. When the sharded query involves one or more joins, however, the shard-local queries typically need to see data from other shards, which requires more processing time and uses more of the memory allocated to the database cache. This extra

overhead can be avoided by enabling a *cosharded join*, in which the rows from the two tables that will be joined are placed on the same shard. When a join is cosharded, a query involving that join is decomposed into shard-local queries that join only rows on the same shard and thus run independently and locally, as with any other sharded query.

You can enable a cosharded join using one of two approaches:

- Specify equivalent UDSKs for two tables.
- Use a SASK for one table and the **coshard with** keywords and the appropriate UDSK with another.

To use equivalent UDSKs, simply specify the frequently joined fields as the shard keys for the two tables. For example, suppose you will be joining the CITATION and VEHICLE tables to return the traffic citations associated with each vehicle, as follows:

```
SELECT * FROM citation, vehicle where citation.vehiclenumber = vehicle.vin
```

To make this join cosharded, you would create both tables with the respective equivalent fields as the shard keys:

```
CREATE TABLE VEHICLE (make VARCHAR(30) not null, model VARCHAR(20) not null,
  year INT not null, vin VARCHAR(17) not null, shard key (vin))

CREATE TABLE CITATION(citationid VARCHAR(8) not null, date DATE not null,
  licensenumber VARCHAR(12) not null, plate VARCHAR(10) not null,
  vehiclenumber VARCHAR(17) not null, shard key (vehiclenumber))
```

Because the sharding algorithm is deterministic, this would result in both the VEHICLE row and the CITATION rows (if any) for a given VIN (a value in the vin and vehiclenumber fields, respectively) being located on the same shard (although the field value itself does not in any way determine which shard each set of rows is on). Thus, when the query cited above is run, each shard-local query can execute the join locally, that is, entirely on its shard. A join cannot be cosharded in this manner unless it includes an equality condition between the two fields used as shard keys. Likewise, you can use multiple-field UDSKs to enable a cosharded join, as long as the shard keys for the respective tables have same number of fields, in the same order, of types that allow the field values to be compared for equality.

The other approach, which is effective in many cases, involves creating one table using a SASK, and then another by specifying the **coshard with** keywords to indicate that it is to be cosharded with the first table, and a shard key with values that are equivalent to the system-assigned RowID values of the first table. For example, you might be frequently joining the ORDER and CUSTOMER tables in queries like the following:

```
SELECT * FROM orders, customers where orders.customer = customers.%ID
```

In this case, because the field on one side of the join represents the RowID, you would start by creating that table, CUSTOMER, with a SASK, as follows:

```
CREATE TABLE CUSTOMER (firstname VARCHAR(50) not null, lastname VARCHAR(75) not null,
  address VARCHAR(50) not null, city VARCHAR(25) not null, zip INT, shard)
```

To enable the cosharded join, you would then shard the ORDER table, in which the customer field is defined as a reference to the CUSTOMER table, by specifying a coshard with the CUSTOMER table on that field, as follows:

```
CREATE TABLE ORDER (date DATE not null, amount DECIMAL(10,2) not null,
  customer CUSTOMER not null, shard key (customer) coshard with CUSTOMER)
```

As with the UDSK example previously described, this would result in each row from ORDER being placed on the same shard as the row from CUSTOMER with RowID value matching its customerid value (for example, all ORDER rows in which customerid=427 would be placed on the same shard as the CUSTOMER row with ID=427). A cosharded join enabled in this manner must include an equality condition between the ID of the SASK-sharded table and the shard key specified for the table that is cosharded with it.

Generally, the most beneficial cosharded joins can be enabled using either of the following, as indicated by your schema:

- SASKs representing structural relationships between tables and the **coshard with** keywords, as illustrated in the example, in which customerid in the ORDER table is a reference to RowID in the CUSTOMER table.
- UDSKs involving semantically meaningful fields that do not correspond to the RowID and so cannot be cosharded using **coshard with**, as illustrated by the use of the equivalent vin and vehiclenumber fields from the VEHICLE and CITATION tables. (UDSKs involving fields that happen to be used in many joins but represent more superficial or adhoc relationships are usually not as helpful.)

Like queries with no joins and those joining sharded and nonsharded data, cosharded joins scale well with increasing numbers of shards, and they also scale well with increasing numbers of joined tables. Joins that are not cosharded perform well with moderate numbers of shards and joined tables, but scale less well with increasing numbers of either. For these reasons, you should carefully consider cosharded joins at this stage, just as, for example, indexing is taken into account to improve performance for frequently-queried sets of fields.

When selecting shard keys, bear in mind these general considerations:

- The shard key of a sharded table cannot be changed, and its values cannot be updated.
- All other things being equal, a balanced distribution of a table's rows across the shards is beneficial for performance, and the algorithms used to distribute rows achieve the best balance when the shard key contains large numbers of different values but no major outliers (in terms of frequency); this is why the default RowID typically works so well. A well-chosen UDSK with similar characteristics may also be effective, but a poor choice of UDSK may lead to an unbalanced data distribution that does not significantly improve performance.
- When a large table is frequently joined to a much smaller one, sharding the large one and making the small one nonsharded may be more effective than enabling a cosharded join.

3.3.2.2 Evaluate Unique Constraints

When a sharded table has a unique constraint (see Field Constraint and Unique Fields Constraint), uniqueness is guaranteed across all shards. Generally, this means uniqueness must be enforced across all shards for each row inserted or updated, which substantially slows insert/update performance. When the shard key is a subset of the fields of the unique key, however, uniqueness can be guaranteed across all shards by enforcing it locally on the shard on which a row is inserted or updated, avoiding any performance impact.

For example, suppose an OFFICES table for a given campus includes the buildingnumber and officenumber fields. While building numbers are unique within the campus, and office numbers are unique within each building, the two must be combined to make each employee's office address unique within the campus, so you might place a unique constraint on the table as follows:

```
CREATE TABLE OFFICES (countrycode CHAR(3), buildingnumber INT not null, officenumber INT not null,
  employee INT not null, CONSTRAINT address UNIQUE (buildingname,officenumber))
```

If the table is to be sharded, however, and you want to avoid any insert/update impact on performance, you must use buildingnumber, officenumber, or both as the shard key. For example, if you shard on buildingnumber (by adding `shard key (buildingnumber)` to the statement above), all rows for each building are located on the same shard, so when inserting a row for the employee whose address is "building 10, office 27", the uniqueness of the address can be enforced locally on the shard containing all rows in which buildingnumber=10; if you shard on officenumber, all rows in which officenumber=27 are on the same shard, so the uniqueness of "building 10, office 27" can be enforced locally on that shard. On the other hand, if you use a SASK, or employee as a UDSK, any combination of buildingnumber and officenumber may appear on any shard, so the uniqueness of "building 10, office 27" must be enforced across all shards, impacting performance.

For these reasons, you may want to avoid defining unique constraints on a sharded table unless one of the following is true:

- All unique constraints are defined with the shard key as a subset (which may not be as effective generally as a SASK or a different UDSK).
- Insert and update performance is considered much less important than query performance for the table in question.

Note: Enforcing uniqueness in application code (for example, based on some counter) can eliminate the need for unique constraints within a table, simplifying shard key selection.

3.3.2.3 Create the Tables

Create the empty sharded tables using standard CREATE TABLE statements (see CREATE TABLE) in the cluster namespace on any data node in the cluster. As shown in the examples in Choose a Shard Key, there are two types of sharding specifications when creating a table:

- To shard on the system-assigned shard key (SASK), include the **shard** keyword in the CREATE TABLE statement.
- To shard on a user-defined shard key (UDSK), follow **shard** with the **key** keyword and the field or fields to shard on, for example **shard key (customerid, purchaseid)**.

Note: If the PK_IS_IDKEY option is set when you create a table, as described in Defining the Primary Key, the table's RowID is the primary key; in such a case, using the default shard key means the primary key is the shard key. The best practice, however, if you want to use the primary key as the shard key, is to explicitly specify the shard key, so that there is no need to determine the state of this setting before creating tables.

You can display a list of all of the sharded tables on a cluster, including their names, owners, and shard keys, by navigating to the Sharding Configuration page of the Management Portal (**System Administration > Configuration > System Configuration > Sharding Configuration**) on node 1 or another data node, selecting the cluster namespace, and selecting the **Sharded Tables** tab. For a table you have loaded with data, you can click the **Details** link to see how many of the table's rows are stored on each data node in the cluster.

Sharded Table Creation Constraints

The following constraints apply to sharded table creation:

- You cannot use ALTER TABLE to make an existing nonsharded table into a sharded table (you can however use ALTER TABLE to alter a sharded table).
- The SHARD KEY fields must be of numeric or string data types. The only collations currently supported for shard key fields are exact, SQLString, and SQLUpper, with no truncation.
- All data types are supported except the ROWVERSION field and SERIAL (%Counter) fields.
- A sharded table cannot include %CLASSPARAMETER *VERSIONPROPERTY*.

For further details on the topics and examples in this section, see CREATE TABLE.

Defining Sharded Tables Using Sharded Classes

In addition to using DDL to define sharded tables, you can define classes as sharded using the Sharded class keyword; for details, see Defining a Sharded Table by Creating a Persistent Class. The class compiler has been extended to warn against using class definition features incompatible with sharding (such as customized storage definitions) at compile time. More developed workload mechanisms and support for some of these incompatible features will be introduced in upcoming versions of InterSystems IRIS.

3.3.3 Load Data Onto the Cluster

Data can be loaded into sharded tables through any InterSystems IRIS interface that supports SQL. Rapid bulk loading of data into sharded tables is supported by the transparent parallel load capability built into the InterSystems IRIS JDBC driver. These options are described in the following:

- You can load data into the empty sharded tables on your cluster through any InterSystems IRIS interface that supports SQL, such as a JDBC client tool or the SQL Shell. The LOAD DATA command, which is intended for rapid population

of a table with well-validated data, loads from a table or file. You can also load data using one or more INSERT statements, including INSERT SELECT FROM.

- The InterSystems IRIS JDBC driver implements specific optimizations for loading sharded tables. When preparing an INSERT statement into a sharded table, the JDBC client automatically opens direct connections to every data node and distributes the inserted rows across them, significantly enhancing performance without requiring any specific configuration or API calls. Any application loading sharded tables using JDBC transparently benefit from this capability.

  Note: For most data loading operations, the JDBC driver uses direct connections to the data nodes brokered by the cluster. This requires the driver client to reach the data nodes at the IP addresses or hostnames with which they were assigned to the cluster, and means you cannot execute such queries if this is not possible.

3.3.4 Create and Load Nonsharded Tables

As with sharded tables, you can create nonsharded tables on any data node, and load data into them, using your customary methods. These tables are immediately available to the cluster for both nonsharded queries and sharded queries that join them to sharded tables. (This is in contrast to architectures in which nonsharded tables must be explicitly replicated to each node that may need them.) See Evaluate Existing Tables for Sharding for guidance in choosing which tables to load as nonsharded.

3.4 Querying the Sharded Cluster

The master namespace and the sharded tables it contains are fully transparent, and SQL queries involving any mix of sharded and nonsharded tables in the master namespace are no different from any SQL queries against any tables in an InterSystems IRIS namespace. No special query syntax is required to identify sharded tables or shard keys. Queries can join multiple sharded tables, as well as sharded and nonsharded tables. Everything is supported except what is specified in the following, which represent limitations and restrictions in the initial version of the InterSystems IRIS sharded cluster; the goal is that they will all be removed.

- The only referential integrity constraints that are enforced for sharded tables are foreign keys when the two tables are cosharded, and the only referential action supported is NO ACTION.
- Shard key fields must be of numeric or string data types. The only collations currently supported for shard key fields are exact, SQLString, and SQLUpper, with no truncation.
- Row-level security for sharded tables is not currently supported.
- Linked tables sourcing their content through a SQL Gateway connection cannot be sharded.
- Use of the following InterSystems IRIS SQL extensions is not currently supported:
  - Aggregate function extensions including %FOREACH, and %AFTERHAVING.
  - Nested aggregate functions.
  - Queries with both an aggregate function and a nonaggregated field in the SELECT or HAVING clauses, unless this nonaggregated field also appears as a GROUP item in the GROUP BY clause.
  - The FOR SOME %ELEMENT predicate condition.
  - The %INORDER keyword.

Note: If you want to explicitly purge cached queries on the data nodes, you can either purge all cached queries from the master namespace, or purge cached queries for a specific table. Both of these actions propagate the purge to the data nodes. Purging of individual cached queries is never propagated to the data nodes.

When queries return time-out errors and/or ECP connection issues are reported in messages.log, these are typically caused by networking issues. Users are advised to call the **$SYSTEM.Sharding.VerifyShards()** method of the %SYSTEM.Sharding API, which will test and try to restore connectivity, reporting some diagnostic information in the process.

When queries report privilege errors, these may be due to a mismatch between user-assigned privileges on the different shards, such as SELECT privileges on a particular table or view. Users are advised to verify and synchronize these on each server. A future version of IRIS will automate this process.

3.5 Additional Sharded Cluster Options

Sharding offers many configurations and options, suitable to your needs. This section provides brief coverage of additional options of interest, including:

- Add data nodes and rebalance data
- Mirror data nodes for high availability
- Deploy compute nodes for workload separation and increased query throughput
- Install multiple data nodes per system

For further assistance in evaluating the benefits of these options for your cluster, please contact the InterSystems Worldwide Response Center (WRC).

3.5.1 Add Data Nodes and Rebalance Data

InterSystems IRIS sharding is designed for scalability and elasticity. As described in Planning an InterSystems IRIS Sharded Cluster, the number of data nodes you include in a cluster when first deployed is influenced by a number of factors, including but not limited to the estimated working set for sharded tables and the compute resources you have available. Over time, however, you may want to add data nodes for a number of reasons, for example because the amount of sharded data on the cluster has grown significantly, or a resource constraint has been removed. Data nodes can be added using any of the deployment methods described in Deploying the Sharded Cluster, automated or manual.

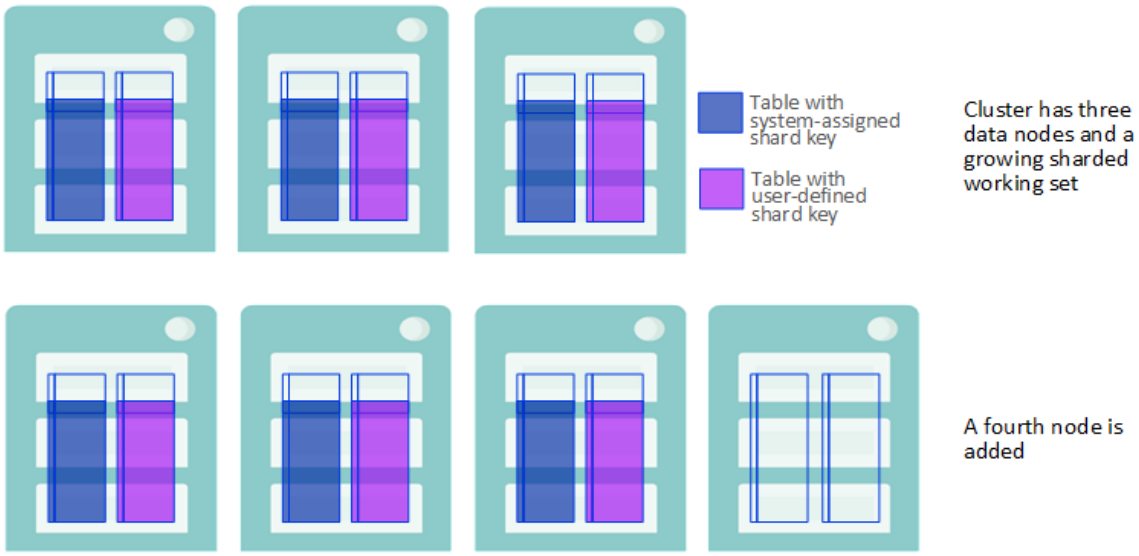
As described in Overview of InterSystems IRIS Sharding, sharding scales query processing throughput by decomposing queries and executing them in parallel on all of the data nodes, with the results merged, aggregated, and returned as full query results to the application. Generally speaking, the greater the amount of sharded data stored on a data node, the longer it will take to return results, and overall query performance is gated by the slowest data node. For optimal performance, therefore, storage of sharded data should be roughly even across the data nodes of the cluster.

This will not be the case after you add data nodes, but *rebalancing* the cluster's sharded data across all of the data nodes restores this roughly even distribution. A cluster can be rebalanced without interrupting the its operation.

The process of adding data nodes and rebalancing data is described in the following example:

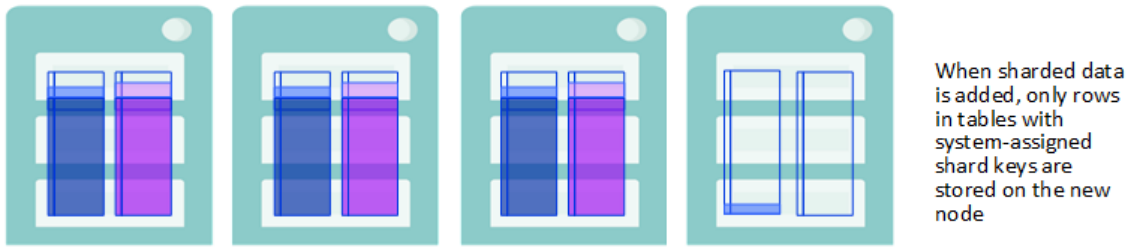
- After you add data nodes to a cluster, the rows of existing sharded tables remain where they were on the original nodes until you rebalance.

Figure 3–2: A Data Node is Added



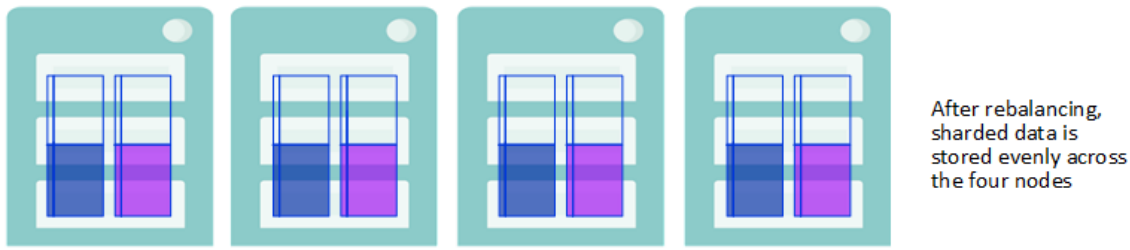
- When sharded data is added to the scaled cluster, in the form of either rows inserted into a previously existing table or new tables created and loaded with data, their storage depends on each table's shard key:
  - If the table has a system-assigned shard key (SASK), the new rows are stored evenly across all of the data nodes, including the new nodes.
  - If the table has a user-defined shard key (UDSK), the new rows are stored evenly across the original set of data nodes only, and not on the newly-added nodes. (If there were no existing UDSK tables before the new nodes were added, however, rows in new UDSK tables are distributed across all data nodes.)

Figure 3–3: New Data is Stored Based on the Shard Key



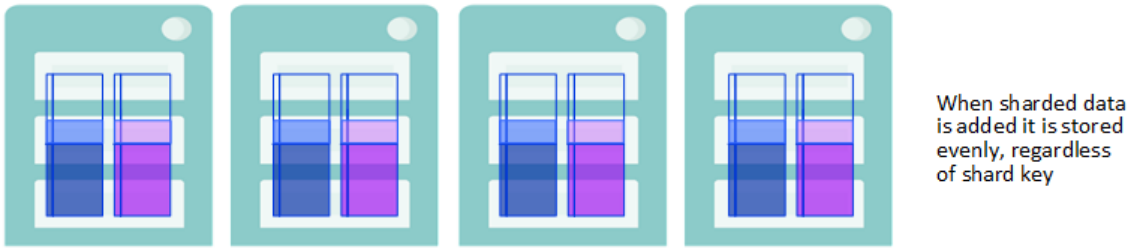
- To take full advantage of parallel query processing on the cluster after adding data nodes, rebalance the sharded data stored on the cluster and enable balanced storage of data added to the cluster in the future.

Figure 3–4: Rebalancing Stores Sharded Data Evenly



- After rebalancing, both rows added to existing tables and the rows of newly created tables are also distributed across all data nodes, regardless of shard key. Thus, once you have rebalanced, all sharded data — in existing tables, rows added to existing tables, and new tables — is evenly distributed across all data nodes.

Figure 3–5: New Sharded Date is Stored Evenly



You can initiate the rebalancing operation in one of two ways:

- Using the REBALANCE dialog, displayed by clicking **Rebalance** on the Management Portal's Configure Node-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Node-Level**).
- Using the **$SYSTEM.Sharding.Rebalance()** API call.

The Class Reference documentation for **$SYSTEM.Sharding.Rebalance()** explains the parameters you can specify using either interface, as well as the detailed mechanics of the rebalancing of sharded data among the data nodes.

All operations on sharded tables are permitted during a rebalancing operation with the exception of JDBC batch inserts to sharded tables (or any bulk load utility that utilizes brokered JDBC batch inserts), which return an error if attempted. An ongoing rebalancing operation may have a small negative effect on query performance and, to a lesser extent, on other operations including updates, inserts, and deletes; creating, altering and dropping tables; and assigning new shards.

If performance is of sufficient concern, you can specify a time limit on the operation, so that it can be scheduled during a low-traffic period or even a maintenance window. When the time limit is reached, the rebalancing operation (if not already complete) stops moving data (although some cleanup tasks may continue for a short time); if there remains data to be rebalanced, you can at the time of your choosing initiate another rebalancing operation, which picks up where the previous one left off. By using this approach you can fully rebalance the cluster with a series of scheduled operations of whatever length suits your needs.

When using the API call, you can also specify the minimum amount of data to be moved by the call; if it is not possible to move that much data within the specified time limit, no rebalancing occurs. Bear in mind that after you have added data nodes, you maximize performance by rebalancing the data.

3.5.2 Mirror Data Nodes for High Availability

An InterSystems IRIS mirror is a logical grouping of physically independent InterSystems IRIS instances simultaneously maintaining exact copies of production databases, so that if the instance providing access to the databases becomes unavailable, another can automatically and quickly take over. This *automatic failover* capability provides high availability for the InterSystems IRIS databases in the mirror. The High Availability Guide contains detailed information about InterSystems IRIS mirroring.

The data nodes in a sharded cluster can be mirrored to give them a failover capability, making them highly available. Each mirrored data node in a sharded cluster includes at least a failover pair of instances, one of which operates as the primary at all times, while the other operates as the backup, ready to take over as primary should its failover partner become unavailable. Data node mirrors in a sharded cluster can also include one or more DR (disaster recovery) async members, which can be promoted to failover member to replace a disabled failover partner or provide disaster recovery if both failover partners become unavailable. For typical configurations, it is strongly recommended that DR asyncs be located in separate data centers or cloud availability zones from their failover pairs to minimize the chances of all of them being affected by the same failure or outage.

A sharded cluster must be either all mirrored or all nonmirrored; that is, mirrored and nonmirrored data nodes cannot be mixed in the same cluster.

The manual procedures for configuring mirrored data nodes (using either the Management Portal or the %SYSTEM.Cluster API) recognize existing mirror configurations. This means you can configure a sharded cluster of mirrored data nodes from either nonmirrored or mirrored instances, as follows, depending on your existing circumstances and requirements::

- When you configure nonmirrored instances as a mirrored sharded cluster, each intended primary you add is automatically configured as a mirror primary before it is attached to the cluster. You can then add another nonmirrored instance as its backup, which is automatically configured as such before it is attached to the cluster.
- When you configure existing mirrors as a sharded cluster, each primary you add is attached to the cluster as a data node primary without any changes to its existing mirror configuration. You can then add its failover partner to the cluster by identifying it as the backup of that primary. (If the mirrored instance you added as a data node primary does not have a failover partner, you can identify a nonmirrored instance as the backup; it is automatically configured as such before it is attached to the cluster.)
- Similarly, you can add a nonmirrored instance as a DR async member of a data node mirror to have it automatically configured as such before being attached to the cluster, or add a DR async member of an existing mirror whose failover members have already been attached to the cluster by identifying it as such.
- In all cases, the globals databases of the cluster and master namespaces (**IRISCLUSTER** and **IRISDM** by default) are added to the mirror (the former on all data nodes and the latter on the node 1 mirror); when you configure existing mirror members, any mirrored databases remain mirrored following sharded cluster configuration.

Generally speaking, the best practice is to either begin with all unmirrored instances and configure the mirrors as part of sharded cluster deployment, or configure all of the data nodes from existing mirrors.

You can deploy a mirrored sharded cluster using any of the methods described in Deploying the Sharded Cluster. The automatic deployment methods described there all include mirroring as an option. This section provides mirrored cluster instructions to replace the final step of the manual procedures included in that section. First, execute the steps described in that section, then use one of the procedures described here to complete deployment, as follows:

1. Plan the data nodes
2. Estimate the database cache and database sizes
3. Provision or identify the infrastructure

   Note: Remember that you must provide two hosts for each mirrored data node planned in the first step. For example, if you plan calls for eight data nodes, your cluster requires 16 hosts. As is recommended for all data nodes in a sharded cluster, the two hosts comprising a mirror should have identical or at least closely comparable specifications and resources.

4. Deploy InterSystems IRIS on the data node hosts
5. Configure the mirrored cluster nodes using either:
   - The Management Portal
   - The %SYSTEM.Cluster API

However you deploy the cluster, the recommended best practices are as follows:

- Load balance application connections across all of the mirrored data nodes in the cluster.
- Deploy and fully configure the mirrored sharded cluster, with all members of all mirrored data nodes (failover pair plus any DR asyncs) attached to the cluster, before any data is stored on the cluster. However, you can also convert an existing nonmirrored cluster to a mirrored cluster, regardless of whether there is data on it.
- To enable transparent query execution after data node mirror failover, include compute nodes in the mirrored cluster.

Note: The Management Portal mirroring pages and the %SYSTEM.MIRROR API allow you to specify more mirror settings than the %SYSTEM.Cluster API and Management Portal sharding pages described here; for details, see Configuring Mirroring. Even if you plan to create a cluster from nonmirrored instances and let the Management Portal or API automatically configure the mirrors, it is a good idea to review the procedures and settings in Creating a Mirror before you do so.

You cannot use the procedures in this section to deploy a mirrored namespace-level cluster. You can, however, deploy a nonmirrored namespace-level cluster as described in Deploying the Namespace-level Architecture and then convert it to a mirrored cluster as described in Convert a Nonmirrored Cluster to a Mirrored Cluster.

3.5.2.1 Mirrored Cluster Considerations

When deploying a mirrored sharded cluster, bear in mind the important points described in the following sections.

Including Compute Nodes in Mirrored Clusters for Transparent Query Execution Across Failover

Because they do not store persistent data, compute nodes are not themselves mirrored, but including them in a mirrored cluster can be advantageous even when the workload involved does not match the advanced use cases described in Deploy Compute Nodes for Workload Separation and Increased Query Throughput (which provides detailed information about compute nodes and procedures for deploying them).

If a mirrored sharded cluster is in asynchronous query mode (the default) and a data node mirror fails over while a sharded query is executing, an error is returned and the application must retry the query. There are two ways to address this problem — that is, to enable sharded queries to execute transparently across failover — as follows:

- Set the cluster to synchronous query mode. This has drawbacks, however; in synchronous mode, sharded queries cannot be canceled, and they make greater use of the **IRISTEMP** database, increasing the risk that it will expand to consume all of its available storage space, interrupting the operation of the cluster.
- Include compute nodes in the cluster. Because the compute node has a mirror connection to the mirrored data node it is assigned to, compute nodes enable transparent query execution across failover in asynchronous mode.

In view of the options, if transparent query execution across failover is important for your workload, InterSystems recommends including compute nodes in your mirrored sharded cluster (there must be at least as many as there are mirrored data nodes). If your circumstances preclude including compute nodes, you can use the RunQueriesAsync option of the **$SYSTEM.Sharding.SetOption()** API call (see %SYSTEM.Sharding API) to change the cluster to synchronous mode, but you should do so only if transparent query execution across failover is more important to you than the ability to cancel sharded queries and manage the size of **IRISTEMP**.

Creating Cluster and Master Namespaces Before Deploying

A sharded cluster is initialized when you configure the first data node, which is referred to as *data node 1*, or simply *node 1*. By default, this includes creating the cluster and master namespaces, named **IRISCLUSTER** and **IRISDM**, respectively, as well as their default globals databases and the needed mappings. However, to control the names of the cluster and master namespaces and/or the characteristics of their globals databases, you can create one or both namespaces and their default databases before configuring the cluster, then specify them during the procedure. If you plan to do this when deploying a mirrored sharded cluster, you cannot begin with unmirrored instances, but instead *must* take the following steps in the order shown:

1. Configure a mirror for each prospective data node, including two failover members in each.
2. Create the intended cluster namespace (using the default name **IRISCLUSTER**, or optionally another name) on each mirror primary, and the intended master namespace (default name **IRISDM**) on the primary of the mirror you will attach as data node 1. When creating each namespace, at the *Select an existing database for Globals* prompt select *Create New Database*, and at the bottom of the second page of the Database Wizard, set *Mirrored database?* to *Yes* to create the namespace's default globals database as a mirrored database.

3. Configure the mirrors as data nodes as described in the procedures in this section, specifying the namespaces you created as the cluster and master namespaces.

Enabling IP Address Override on All Mirror Members

In some cases the hostname known to the InterSystems IRIS instance on an intended cluster node does not resolve to an appropriate address, or no hostname is available. If for this or any other reason you want other cluster nodes to communicate with a node using its IP address instead, you can enable this by providing the node's IP address, at the **Override hostname with IP address** prompt in the Management Portal or as an argument to the **$SYSTEM.Cluster.InitializeMirrored()** and **$SYSTEM.Cluster.AttachAsMirroredNode()** calls. When you do this for one member of a mirror, you must do it for all mirror members, as follows:

- When configuring unmirrored instances as mirrored data nodes, ensure that you enable IP address override for all mirror members using the prompt or calls cited above.
- When configuring an existing mirror as a data node, however, you must enable IP address override on all members of the mirror *before* adding it to the cluster. Regardless of whether you are using the Management Portal or API procedure, do this on each node by opening an InterSystems Terminal window and (in any namespace) calling **$SYSTEM.Sharding.SetNodeIPAddress()** (see %SYSTEM.Sharding API), for example:

  ```
  set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
  ```

  Once you have used this call on a node, you must use the IP address you specified, rather than the hostname, to refer to the node in other API calls, for example, when attaching a mirror backup to the cluster using the **$SYSTEM.Cluster.AttachAsMirroredNode()** call, in which you must identify the attached primary.

Updating a Mirrored Cluster's Metadata

If you make mirroring changes to the mirrored data nodes of a cluster outside of the %SYSTEM.Cluster API and Management Portal sharding pages, for example using the Management Portal mirroring pages or the SYS.Mirror API, you must update the mirrored cluster's metadata after doing so; for more information, see Updating the Cluster Metadata for Mirroring Changes.

3.5.2.2 Configure the Mirrored Cluster Using the Management Portal

For information about opening the Management Portal in your browser, see the instructions for an instance deployed in a container or one installed from a kit.

To configure the mirrored cluster using the Management Portal, follow these steps:

1. On both the intended node 1 primary and the intended node 1 backup, open the Management Portal for the instance, select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK** (here is no need to change the **Maximum Number of ECP Connections** setting). Then restart the instance as indicated (there is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted).
2. On the intended primary, navigate to the Configure Node-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Node-Level**) and click the **Configure** button.
3. On the CONFIGURE NODE-LEVEL CLUSTER dialog, select **Initialize a new sharded cluster on this instance** and respond to the prompts that display as follows:
   a. Select a **Cluster namespace** and **Master namespace** from the respective drop-downs, which both include
      - The default names (**IRISCLUSTER** and **IRISDM**) of new namespaces to be created, along with their default globals databases.
      - All eligible existing namespaces.

Initializing a sharded cluster creates by default cluster and master namespaces named **IRISCLUSTER** and **IRISDM**, respectively, as well as their default globals databases and the needed mappings. However, to control the names of the cluster and master namespaces and the characteristics of their globals databases, you can create one or both namespaces and their default databases before configuring the cluster, then specify them during the procedure. For example, given the considerations discussed in Globals Database Sizes, you may want to do this to control the characteristics of the default globals database of the cluster namespace, or *shard database*, which is replicated on all data nodes in the cluster.

Note: If you want to use existing namespaces when configuring existing mirrors as a sharded cluster, follow the procedure in Creating Cluster and Master Namespaces Before Deploying.

If the default globals database of an existing namespace you specify as the cluster namespace contains any globals or routines, initialization will fail with an error.

b. In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, enter the IP address at the **Override hostname** prompt.

c. Select **Enable Mirroring**, and add the arbiter's location and port if you intend to configure one (which is a highly recommended best practice).

4. Click **OK** to return to the Configure Node-Level page, which now includes two tabs, **Shards** and **Sharded Tables**. Node 1 is listed under **Shards**, including its cluster address (which you will need in the next procedure), so you may want to leave the Management Portal for node 1 open on the Configure Node-Level page, for reference. The mirror name is not yet displayed because the backup has not yet been added.
5. On the intended node 1 backup, navigate to the Configure Node-Level page as for the primary, and click the **Configure** button.
6. On the CONFIGURE NODE-LEVEL CLUSTER dialog, select **Add this instance to an existing sharded cluster** and respond to the prompts that display as follows:

   a. Enter the address displayed for the node 1 primary on the **Shards** tab of the Configure Node-Level page (as described in an earlier step) as the **Cluster URL**.

   b. Select **data** at the **Role** prompt to configure the instance as a data node.

   c. In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, enter the IP address at the **Override hostname** prompt.

   d. Select **Mirrored cluster** and do the following:

   - Select **backup failover** from the **Mirror role** drop-down.
   - If you configured an arbiter when initializing node 1 in a previous step, add the same arbiter location and port as you did there.

7. Click **OK** to return to the Configure Node-Level page, which now includes two tabs, **Shards** and **Sharded Tables**. The node 1 primary and backup you have configured so far are listed under **Shards** in the node 1 position, with the assigned mirror name included.
8. For each remaining mirrored data node, repeat the previous steps, beginning with the **Enable Sharding** option and a restart for both instances. When you add the primary to the cluster, enter the node 1 primary's address as the **Cluster URL**, as described in the preceding, but when you add the backup, enter the address of the primary you just added as the **Cluster URL** (not the address of the node 1 primary).
9. When each mirrored data node has been added, on one of the primaries in the cluster, navigate to the Configure Node-Level page and click **Verify Shards** to verify that the new mirrored node is correctly configured and can communicate

with the others. You can also wait until you have added all the mirrored data nodes to do this, or you can make the verification operation automatic by clicking the **Advanced Settings** button and selecting **Automatically verify shards on assignment** on the ADVANCED SETTINGS dialog. (Other settings in this dialog should be left at the defaults when you deploy a sharded cluster.)

3.5.2.3 Configure the Mirrored Cluster Nodes Using the %SYSTEM.Cluster API

To configure the mirrored cluster using the API, do the following:

1. On the intended node 1 primary, open the InterSystems Terminal for the instance and call the **$SYSTEM.Cluster.InitializeMirrored()** method, for example:

```
set status = $SYSTEM.Cluster.InitializeMirrored()
```

Note: To see the return value (for example, 1 for success) for the each API call detailed in these instructions, enter:

```
zw status
```

If a call does not succeed, display the user-friendly error message by entering:

```
do $SYSTEM.Status.DisplayError(status)
```

This call initializes the cluster on the node in the same way as **$SYSTEM.Cluster.Initialize()**, described in Configure Node 1; review that section for explanations of the first four arguments (none required) to **InitializeMirrored()**, which are the same as for **Initialize()**. If the instance is not already a mirror primary, you can use the next five arguments to configure it as one; if it is already a primary, these are ignored. The mirror arguments are as follows:

- Arbiter host
- Arbiter port
- Directory containing the Certificate Authority certificate, local certificate, and private key file required to secure the mirror with TLS, if desired. The call expects the files to be named CAFile.pem, CertificateFile.pem, and PrivateKeyFile.pem, respectively.
- Name of the mirror.
- Name of this mirror member.

Note: The **InitializeMirrored()** call returns an error if

- The current InterSystems IRIS instance is already a node of a sharded cluster.
- The current instance is already a mirror member, but not the primary.
- You specify (in the first two arguments) a cluster namespace or master namespace that already exists, and its globals database is not mirrored.

2. On the intended node 1 backup, open the Terminal for the InterSystems IRIS instance and call **$SYSTEM.Cluster.AttachAsMirroredNode()**, specifying the host and superserver port of the node 1 primary as the cluster URL in the first argument, and the mirror role **backup** in the second, for example:

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://node1prim:1972","backup")
```

If you supplied an IP address as the fourth argument to **InitializeMirrored()** when initializing the node 1 primary, use the IP address instead of the hostname to identify node 1 in the first argument, for example:

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://100.00.0.01:1972","backup")
```

Note: The default superserver port number of a noncontainerized InterSystems IRIS instance that is the only such on its host is 1972. To see or set the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal. (For information about opening the Management Portal for the instance and determining the superserver port, see the instructions for an instance deployed in a container or one installed from a kit.)

This call attaches the node as a data node in the same way as **$SYSTEM.Cluster.AttachAsDataNode()**, as described in Configure the Remaining Data Nodes, and ensures that it is the backup member of the node 1 mirror. If the node is backup to the node 1 primary before you issue the call — that is, you are initializing an existing mirror as node 1 — the mirror configuration is unchanged; if it is not a mirror member, it is added to the node 1 primary's mirror as backup. Either way, the namespace, database, and mappings configuration of the node 1 primary are replicated on this node. (The third argument to **AttachAsMirroredNode** is the same as the second for **AttachAsDataNode**, that is, the IP address of the host, included if you want the other cluster members to use it in communicating with this node.)

If there are any intended DR async members of the node 1 mirror, use **AttachAsMirroredNode()** to attach them, with the substitution of **drasync** for **backup** as the second argument, for example:

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://node1prim:1972","drasync")
```

As with attaching a backup, if you are attaching an existing member of the mirror, its mirror configuration is unchanged; otherwise, the needed mirror configuration is added. Either way, the namespace, database, and mappings configuration of the node 1 primary are replicated on the new node.

Note: Attempting to attach an instance that is a member of a different mirror from that of the node 1 primary causes an error.

3. To configure mirrored data nodes other than node 1, use **$SYSTEM.Cluster.AttachAsMirroredNode()** to attach both the failover pair and any DR asyncs to the cluster, as follows:

   a. When adding a primary, specify any existing primary in the cluster URL and **primary** as the second argument. If the instance is not already the primary in a mirror, use the fourth argument and the four that follow to configure it as the first member of a new mirror; the arguments are as listed for the **InitializeMirrored()** call in the preceding. If the instance is already a mirror primary, the mirror arguments are ignored if provided.

   b. When adding a backup, specify its intended primary in the cluster URL and **backup** as the second argument. If the instance is already configured as backup in the mirror in which the node you specify is primary, its mirror configuration is unchanged; if it is not yet a mirror member, it is configured as the second failover member.

   c. When adding a DR async, specify its intended primary in the cluster URL and **drasync** as the second argument. If the instance is already configured as a DR async in the mirror in which the node you specify is primary, its mirror configuration is unchanged; if it is not yet a mirror member, it is configured as a DR async.

Note: The **AttachAsMirroredNode()** call returns an error if

- The current InterSystems IRIS instance is already a node in a sharded cluster.
- The role **primary** is specified and the cluster node specified in the cluster URL (first argument) is not a mirror primary, or the current instance belongs to a mirror in a role other than primary.
- The role **backup** is specified and the cluster node specified in the first argument is not a mirror primary, or is primary in a mirror that already has a backup failover member.
- The role **drasync** is specified and the cluster node specified in the first argument is not a mirror primary.
- The role **backup** or **drasync** is specified and the instance being added already belongs to a mirror other than the one whose primary you specified.
- The cluster namespace (or master namespace, when adding the node 1 backup) already exists on the current instance and its globals database is not mirrored.

4. When you have configured all of the data nodes, you can call the **$SYSTEM.Cluster.ListNodes()** method to list them. When a cluster is mirrored, the list indicates the mirror name and role for each member of a mirrored data node, for example:

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    Host          Port   Mirror  Role
1       Data        node1prim     1972   MIRROR1 Primary
1       Data        node1back     1972   MIRROR1 Backup
1       Data        node1dr       1972   MIRROR2 DRasync
2       Data        node2prim     1972   MIRROR2 Primary
2       Data        node2back     1972   MIRROR2 Backup
2       Data        node2dr       1972   MIRROR2 DRasync
```

3.5.2.4 Convert a Nonmirrored Cluster to a Mirrored Cluster

This section provides a procedure for converting an existing nonmirrored sharded cluster to a mirrored cluster. The following is an overview of the tasks involved:

- Provision and prepare at least enough new nodes to provide a backup for each existing data node in the cluster.
- Create a mirror on each existing data node and then call **$SYSTEM.Sharding.AddDatabasesToMirrors()** on node 1 to automatically convert the cluster to a mirrored configuration.
- Create a coordinated backup of the now-mirrored master and shard databases on the existing data nodes (the first failover member in each mirror) as described in Coordinated Backup and Restore of Sharded Clusters.
- For each intended second failover member (new node), select the first failover member (existing data node) to be joined, then create databases on the new node corresponding to the mirrored databases on the first failover member, add the new node to the mirror as second failover member, and restore the databases from the backup made on the first failover member to automatically add them to the mirror.
- To add a DR async to the failover pair you have created in a data node mirror, create databases on the new node corresponding to the mirrored databases on the first failover member, add the new node to the mirror as a DR async, and restore the databases from the backup made on the first failover member to automatically add them to the mirror.
- Call **$SYSTEM.Sharding.VerifyShards()** on any of the mirror primaries (original data nodes) to validate information about the backups and add it to the sharding metadata.

You can perform the entire procedure within a single maintenance window (that is, a scheduled period of time during which the application is offline and there is no user activity on the cluster), or you can split it between two maintenance windows, as noted in the instructions.

The detailed steps are provided in the following. If you are not already familiar with it, review Deploy the Cluster Using the %SYSTEM.Cluster API before continuing. Familiarity with mirror configuration procedures, as described in Configuring Mirroring, is also helpful but not required; the steps in this procedure provide links where appropriate.

Important: When a node-level nonmirrored cluster is converted to mirrored using this procedure, it becomes a namespace-level cluster, and can be managed and modified using only the %SYSTEM.Sharding API and the namespace-level pages in the Management Portal.

1. To use this procedure, you must know the names of the cluster's cluster namespace and master namespace, which were determined when you deployed the cluster. For example, step 4 in Configure Data Node 1 discusses selecting the cluster and master namespaces; similarly, the discussion of the initial API call in Configure Node 1 includes determining the cluster and master namespaces.
2. Prepare the nodes that are to be added to the cluster as backup failover members according to the instructions in Provision or identify the infrastructure and Deploy InterSystems IRIS on the Data Nodes. The host characteristics and InterSystems IRIS configuration of the prospective backups should be the same as the existing data nodes in all respects (see Mirror Configuration Guidelines).

   Note: It may be helpful to make a record, by hostnames or IP addresses, of the intended first failover member (existing data node) and second failover member (newly added node) of each failover pair.
3. Begin a maintenance window for the sharded cluster.
4. On each current data node, start the ISCAgent, then create a mirror and configure the first failover member.
5. To convert the cluster to a mirrored configuration — that is, to incorporate the mirrors you created in the previous step into the cluster's configuration and metadata — open the InterSystems Terminal for the instance on node 1 and in the master namespace call the **$SYSTEM.Sharding.AddDatabasesToMirrors()** method (see %SYSTEM.Sharding API) as follows:

   ```
   set status = $SYSTEM.Sharding.AddDatabasesToMirrors()
   ```

   Note: To see the return value (for example, 1 for success) for the each API call detailed in these instructions, enter:

   ```
   zw status
   ```

   Reviewing **status** after each call is a good general practice, as a call might fail silently under some circumstances. If a call does not succeed (**status** is not **1**), display the user-friendly error message by entering:

   ```
   do $SYSTEM.Status.DisplayError(status)
   ```

   The **AddDatabasesToMirrors()** call does the following:

   - Adds the master and shard databases on node 1 (see Initialize node 1) and the shard databases on the other data nodes to their respective mirrors.
   - Reconfigures all ECP connections between nodes as mirror connections, including those between compute nodes (if any) and their associated data nodes.
   - Reconfigures remote databases on all data nodes and adjusts all related mappings accordingly.
   - Updates the sharding metadata to reflect the reconfigured connections, databases, and mappings.

   When the call has successfully completed, the sharded cluster is in a fully usable state (although failover is not yet possible because the backup failover members have not yet been added).
6. Perform a coordinated backup of the data nodes (that is, one in which all nodes are backed up at the same logical point in time). Specifically, on each of the first failover members (the existing data nodes), back up the shard database, and

on node 1, also back up the master database. Before the backup, confirm that you have identified the right databases by examining the instance's configuration parameter file (CPF), as follows:

- Identify the shard database by finding the `[Map.clusternamespace]` section of the CPF — for example, if the cluster namespace is **CLUSTERNAMESPACE**, the section would be `[Map.CLUSTERNAMESPACE]` — and locating the **IRIS.SM.Shard** and **IS.*** global prefix mappings, the target of which is the shard database. Additional global prefixes may be mapped to the shard database, as shown in the following, which identifies the shard database as **SHARDDB**.

```
[Map.CLUSTERNAMESPACE]
Global_IRIS.SM.Shard=SHARDDB
Global_IRIS.Shard.*=SHARDDB
Global_IS.*=SHARDDB
Package_IRIS.Federated=SHARDDB
```

- On node 1, also locate the `[Namespaces]` section, where the master database is shown after the master namespace as its default globals database. For example, the following shows **MASTERDB**, the master database, as the default globals databases of **MASTERNAMESPACE**.

```
[Namespaces]
%SYS=IRISSYS
CLUSTERNAMESPACE=SHARDDB
MASTERNAMESPACE=MASTERDB
USER=USER
```

7. Optionally, end the current maintenance window and allow application activity while you prepare the prospective second failover members and DR async mirror members (if any) in the following steps.
8. On each node to be added to the cluster as a second failover member or DR async:
   a. Start the ISCAgent.
   b. Create a namespace with the same name as the cluster namespace on the intended first failover member (**CLUSTERNAMESPACE** in the examples above), configuring as its default globals database a local database with the same name as the shard database on the first failover member (**SHARDDB** in the examples).
   c. On the intended second failover member or DR async to be added to the node 1, also create a namespace with the same name as the master namespace on the intended first failover member, configuring as its default globals database a local database with the same name as the master database. Using the examples above, you would create a **MASTERNAMESPACE** namespace with a database called **MASTERDB** as its default globals database.
9. If not in a maintenance window, start a new one.
10. On each new node, perform the tasks required to add any nonmirrored instance as the second failover member or a DR async member of an existing mirror that includes mirrored databases containing data, as follows:
    - Configure the node as the second failover member of the intended mirror or (after the second failover member has been configured) configure it as a DR async member.
    - On the newly configured member, restore the shard database from the backup made on the first failover member; on a newly configured member of the node 1 mirror (second failover or DR async), also restore the master database from the backup made on the node 1 first failover member.
    - Activate and catch up the master database and the cluster databases (not necessary if you created the backups using online backup).

Note: Sharding automatically updates the cluster namespace definition, creates all the needed mappings, ECP server definitions, and remote database definitions, and propagates to the shards any user-defined mappings in the master namespace. Therefore, the only mappings that must be manually created during this process are any user-defined mappings in the master namespace, which must be created only in the master namespace on the node 1 second failover member. ECP server definitions and remote database definitions need not be manually copied.

11. Open the InterSystems Terminal for the instance on any of the primaries (original data nodes) and in the cluster namespace (or the master namespace on node 1) call the **$SYSTEM.Sharding.VerifyShards()** method (see %SYSTEM.Sharding API) as follows:

    ```
    set status = $SYSTEM.Sharding.VerifyShards()
    ```

    This call automatically adds the necessary information about the second failover members of the mirrors to the sharding metadata.

    Note: All of the original cluster nodes must be the current primary of their mirrors when this call is made. Therefore, if any mirror has failed over since the second failover member was added, arrange a planned failover back to the original failover member before performing this step. (For one procedure for planned failover, see Maintenance of Primary Failover Member; for information about using the **iris stop** command for the graceful shutdown referred to in that procedure, see Controlling InterSystems IRIS Instances.)

Important: With the completion of the last step above, the maintenance window can be terminated. However, InterSystems strongly recommends testing each mirror by executing a planned failover (see above) before the cluster goes into production.

3.5.2.5 Updating the Cluster Metadata for Mirroring Changes

When you make changes to the mirror configuration of one or more data nodes in a mirrored cluster using any means other than the API or Management Portal procedures described here — that is, using the Mirroring pages of the Management Portal, the **^MIRROR** routine, or the SYS.MIRROR API) — you must update the cluster's metadata by either calling **$SYSTEM.Sharding.VerifyShards()** (see %SYSTEM.Sharding API) in the cluster namespace or using the **Verify Shards** button on the Configure Node-Level page of the Management Portal (see Configure the Mirrored Cluster Using the Management Portal) on any current primary failover member in the cluster. For example, if you perform a planned failover, add a DR async, demote a backup member to DR async, or promote a DR async to failover member, verifying the shards updates the metadata to reflect the change. Updating the cluster metadata is an important element in maintaining and utilizing a disaster recovery capability you have established by including DR asyncs in your data node mirrors; for more information, see Disaster Recovery of Mirrored Sharded Clusters.

A cluster's shards can be verified after every mirroring configuration operation, or just once after a sequence of operations, but if operations are performed while the cluster is online, it is advisable to verify the shards immediately after any operation which adds or removes a failover member.

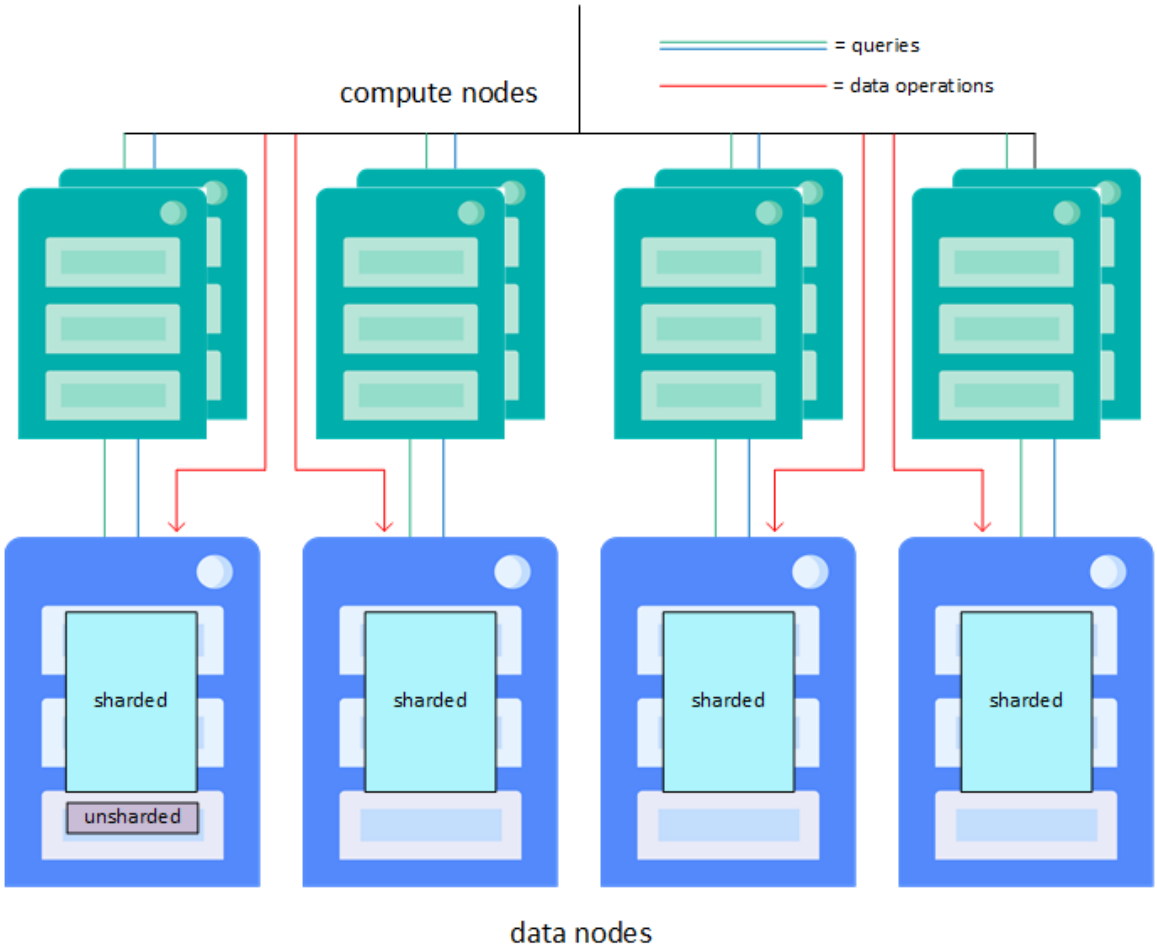
3.5.3 Deploy Compute Nodes for Workload Separation and Increased Query Throughput

For advanced use cases in which extremely low query latencies are required, potentially at odds with a constant influx of data, *compute nodes* can be added to provide a transparent caching layer for servicing queries. Each compute node caches the sharded data on the data node it is associated with, as well as nonsharded data when necessary. When a cluster includes compute nodes, read-only queries are automatically executed in parallel on the compute nodes, rather than on the data nodes; all write operations (insert, update, delete, and DDL operations) continue to be executed on the data nodes. This division of labor separates the query and data ingestion workloads while maintaining the advantages of parallel processing

and distributed caching, improving the performance of both. Assigning multiple compute nodes per data node can further improve the query throughput and performance of the cluster.

When compute nodes are added to a cluster, they are automatically distributed as evenly as possible across the data nodes. Adding compute nodes yields significant performance improvement only when there is at least one compute node per data node; since a cluster is only as fast as its slowest data node, the most efficient use of resources, in general, is to assign the same number of compute nodes to each data node. Because compute nodes support query execution only and do not store any data, their hardware profile can be tailored to suit those needs, for example by emphasizing memory and CPU and keeping storage to the bare minimum.

Figure 3–6: Sharded cluster with compute nodes



For information about planning compute nodes and load balancing application connections to clusters with compute nodes, see Plan Compute Nodes.

You can add compute nodes to a sharded cluster using any of the deployment methods described in Deploying the Sharded Cluster. The automatic deployment methods described there all include compute nodes as an option. This section provides additional manual instructions for deploying compute nodes. First, manually deploy and configure the sharded cluster using the steps described in that section, then complete deployment with the steps described here, as follows:

1. Plan data nodes
2. Estimate the database cache and database sizes
3. Provision or identify the infrastructure

Note: Include hosts for your planned compute nodes along with those for the planned data nodes. For example, if you plan calls for eight data nodes and eight compute nodes, your cluster requires 16 hosts. All nodes in a sharded cluster should have identical or at least closely comparable specifications and resources, with the exception of storage for compute nodes, which do not use any in their sharded cluster role.

4. Deploy InterSystems IRIS on the data node hosts
5. Configure the data nodes using either the Management Portal or the %SYSTEM.Cluster API.
6. Add compute nodes using either the Management Portal or the %SYSTEM.Cluster API, as described in the following sections.

3.5.3.1 Configure or Deploy Compute Nodes Using the Management Portal

When you add a compute node to a cluster, it is assigned to a data node that previously had the minimum number of associated compute nodes, so as to automatically balance compute nodes across the data nodes. The procedure is the same regardless of whether the cluster is mirrored.

For information about opening the Management Portal in your browser, see the instructions for an instance deployed in a container or one installed from a kit.

To add an instance on a networked system to your cluster as a compute node, use the following steps.

1. Open the Management Portal for the instance, select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK**. (The value of the **Maximum Number of ECP Connections** setting need not be changed as the default is appropriate for virtually all clusters.)
2. Restart the instance. (There is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted.)
3. Navigate to the Configure Node-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Node-Level**) and click the **Configure** button.
4. On the CONFIGURE NODE-LEVEL CLUSTER dialog, select **Add this instance to an existing sharded cluster** and respond to the prompts that display as follows:
   - Enter the cluster URL, which is the address displayed for any node on the **Shards** tab of the Configure Node-Level page on an instance that already belongs to the cluster, as described in Configure the remaining data nodes.

     Note: If the cluster is mirrored, enter the address of a primary data node or a compute node, but not a backup data node.

   - Select **compute** at the **Role** prompt to configure the instance as a data node.
   - In some cases, the hostname known to InterSystems IRIS does not resolve to an appropriate address, or no hostname is available. If for this or any other reason, you want other cluster nodes to communicate with this node using its IP address instead, enter the IP address at the **Override hostname** prompt.
   - The **Mirrored cluster** check box is not available, because configuration of compute nodes is the same regardless of mirroring (except for the cluster URL provided, as above.)
5. Click **OK** to return to the Configure Node-Level page, which now includes two tabs, **Shards** and **Sharded Tables**. The data and compute nodes you have configured so far are listed under **Shards**, starting with node 1, and showing which data node each compute node is assigned to.

   Click **Verify Shards** to verify that the compute node is correctly configured and can communicate with the others.

Note: If you have many compute nodes to configure, you can make the verification operation automatic by clicking the **Advanced Settings** button and selecting **Automatically verify shards on assignment** on the ADVANCED SETTINGS dialog. (Other settings in this dialog should be left at the defaults when you deploy a sharded cluster.)

3.5.3.2 Deploy Compute Nodes Using the %SYSTEM.Cluster API

To add an instance on a networked system to your cluster as a compute node, open the InterSystems Terminal for the instance and call the **$SYSTEM.Cluster.AttachAsComputeNode()** method specifying the hostname of an existing cluster node and the superserver port of its InterSystems IRIS instance, for example:

```
set status = $SYSTEM.Cluster.AttachAsComputeNode("IRIS://datanode2:1972")
```

Note: To see the return value (for example, 1 for success) for the each API call detailed in these instructions, enter:

```
zw status
```

If a call does not succeed, display the user-friendly error message by entering:

```
do $SYSTEM.Status.DisplayError(status)
```

If you provided the IP address of the template node when configuring it (see Configure node 1), use the IP address instead of the hostname.

```
set status = $SYSTEM.Cluster.AttachAsComputeNode("IRIS://100.00.0.01:1972")
```

If you want other nodes to communicate with this one using its IP address, specify the IP address as the second argument.

Note: From the perspective of another node (which is what you need in this procedure), the superserver port of a containerized InterSystems IRIS instance depends on which host port the superserver port was published or exposed as when the container was created. For details on and examples of this, see Running an InterSystems IRIS Container with Durable %SYS and Running an InterSystems IRIS Container: Docker Compose Example and see Container networking in the Docker documentation.

The default superserver port number of a noncontainerized InterSystems IRIS instance that is the only such on its host is 1972. To see or set the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal. (For information about opening the Management Portal for the instance and determining the superserver port, see the instructions for an instance deployed in a container or one installed from a kit.)

If the cluster node you identify in the first argument is a data node, it is used as the template; if it is a compute node, the data node to which it is assigned is used as the template. The **AttachAsComputeNode()** call does the following:

- Enables the ECP and sharding services
- Associates the new compute node with a data node that previously had the minimum number of associated compute nodes, so as to automatically balance compute nodes across the data nodes.
- Creates the cluster namespace, configuring it to match the settings on the template node (specified in the first argument), as described in Configure Node 1, and creating all needed mappings.
- Sets all SQL configuration options to match the template node.

If a namespace of the same name as the cluster namespace already exists on the new compute node, it is used as the cluster namespace, and only the mappings are replicated.

If you want other cluster nodes to communicate with this node using its IP address instead of its hostname, supply the IP address as the second argument.

The **AttachAsComputeNode()** call returns an error if the InterSystems IRIS instance is already a node in a sharded cluster.

When you have configured all of the compute nodes, you can call the **$SYSTEM.Cluster.ListNodes()** method to list them, for example:

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    DataNodeId    Host          Port
1       Data                      datanode1     1972
2       Data                      datanode2     1972
3       Data                      datanode3     1972
1001    Compute     1             computenode1  1972
1002    Compute     2             computenode2  1972
1003    Compute     3             computenode3  1972
```

When compute nodes are deployed, the list indicates the node ID of the data node that each compute node is assigned to. You can also use the **$SYSTEM.Cluster.GetMetadata()** retrieve metadata for the cluster, including the names of the cluster and master namespaces and their default globals databases and settings for the node on which you issue the call.

3.5.4 Install Multiple Data Nodes per Host

With a given number of systems hosting data nodes, configuring multiple data node instances per host using the %SYSTEM.Sharding API (you cannot do so with the %SYSTEM.Cluster API or the Management Portal) can significantly increase data ingestion throughput. Therefore, when achieving the highest data ingestion throughput at the lowest cost is a concern, this may be achieved by configuring two or three data node instances per host. The gain achieved will depend on server type, server resources, and overall workload. While adding to the total number of systems might achieve the same throughput gain, or more (without dividing a host system's memory among multiple database caches), adding instances is less expensive than adding systems.

3.6 InterSystems IRIS Sharding Reference

This section contains additional information about planning, deploying, and using a sharded configuration, including the following:

- Planning an InterSystems IRIS Sharded Cluster
- Coordinated Backup and Restore of Sharded Clusters
- Disaster Recovery of Mirrored Sharded Clusters
- Cloning a Sharded Cluster
- Sharding APIs
- Deploying the Namespace-level Architecture
- Reserved Names

3.6.1 Planning an InterSystems IRIS Sharded Cluster

This section provides some first-order guidelines for planning a basic sharded cluster, and for adding compute nodes if appropriate. It is not intended to represent detailed instructions for a full-fledged design and planning process. The following tasks are addressed:

- Combine sharding with vertical scaling

- Plan a basic cluster of data nodes
- Plan compute nodes

3.6.1.1 Combine Sharding with Vertical Scaling

Planning for sharding typically involves considering the tradeoff between resources per system and number of systems in use. At the extremes, the two main approaches can be stated as follows:

- Scale vertically to make each system and instance as powerful as feasible, then scale horizontally by adding additional powerful nodes.
- Scale horizontally using multiple affordable but less powerful systems as a cost-effective alternative to one high-end, heavily-configured system.

In practice, in most situations, a combination of these approaches works best. Unlike other horizontal scaling approaches, InterSystems IRIS sharding is easily combined with InterSystems IRIS's considerable vertical scaling capacities. In many cases, a cluster hosted on reasonably high-capacity systems with a range of from 4 to 16 data nodes will yield the greatest benefit.

3.6.1.2 Plan a Basic Cluster of Data Nodes

To use these guidelines, you need to estimate several variables related to the amount of data to be stored on the cluster.

1. First, review the data you intend to store on the cluster to estimate the following:
   a. Total size of all the sharded tables to be stored on the cluster, including their indexes.
   b. Total size of the nonsharded tables (including indexes) to be stored on the cluster that will be frequently joined with sharded tables.
   c. Total size of all of the nonsharded tables (including indexes) to be stored on the cluster. (Note that the previous estimate is a subset of this estimate.)
2. Translate these totals into estimated working sets, based on the proportion of the data that is regularly queried.

   Estimating working sets can be a complex matter. You may be able to derive useful information about these working sets from historical usage statistics for your existing database cache(s). In addition to or in place of that, divide your tables into the three categories and determine a rough working set for each by doing the following:

   - For significant SELECT statements frequently made against the table, examine the WHERE clauses. Do they typically look at a subset of the data that you might be able to estimate the size of based on table and column statistics? Do the subsets retrieved by different SELECT statements overlap with each other or are they additive?
   - Review significant INSERT statements for size and frequency. It may be more difficult to translate these into working set, but as a simplified approach, you might estimate the average hourly ingestion rate in MB (records per second * average record size * 3600) and add that to the working set for the table.
   - Consider any other frequent queries for which you may be able to specifically estimate results returned.
   - Bear in mind that while queries joining a nonsharded table and a sharded table count towards the working set *NonshardSizeJoinedWS*, queries against that same nonsharded data table that do not join it to a sharded table count towards the working set *NonshardSizeTotalWS*; the same nonsharded data can be returned by both types of queries, and thus would count towards both working sets.

   You can then add these estimates together to form a single estimate for the working set of each table, and add those estimates to roughly calculate the overall working sets. These overall estimates are likely to be fairly rough and may turn out to need adjustment in production. Add a safety factor of 50% to each estimate, and then record the final total data sizes and working sets as the following variables:

Table 3–1: Cluster Planning Variables

Variable	Value
ShardSize, *ShardSizeWS*	Total size and working set of sharded tables (plus safety factor)
NonshardSizeJoined, *NonshardSizeJoinedWS*	Total size and working set of nonsharded tables that are frequently joined to sharded tables (plus safety factor)
NonshardSizeTotal, *NonshardSizeTotalWS*	Total size and working set of nonsharded tables (plus safety factor)
NodeCount	Number of data node instances

In reviewing the guidelines in the table that follows, bear the following in mind:

- Generally speaking and all else being equal, more shards will perform faster due to the added parallelism, up to a point of diminishing returns due to overhead, which typically occurs at around 16 data nodes.
- The provided guidelines represent the ideal or most advantageous configuration, rather than the minimum requirement.

  For example, as noted in Evaluating the Benefits of Sharding, sharding improves performance in part by caching data across multiple systems, rather than all data being cached by a single nonsharded instance, and the gain is greatest when the data in regular use is too big to fit in the database cache of a nonsharded instance. As indicated in the guidelines, for best performance the database cache of each data node instance in a cluster would equal at least the combined size of the sharded data working set and the frequently joined nonsharded data working set, with performance decreasing as total cache size decreases (all else being equal). But as long as the total of all the data node caches is greater than or equal to the cache size of a given single nonsharded instance, the sharded cluster will outperform that nonsharded instance. Therefore, if it is not possible to allocate database cache memory on the data nodes equal to what is recommended, for example, get as close as you can. Furthermore, your initial estimates may turn out to need adjustment in practice.
- *Database cache* refers to the database cache (global buffer pool) memory allocation that must be made for each instance. You can allocate the database cache on your instances as follows:
  - When using one of the automated deployment methods, you can override the default globals setting as part of deployment.
  - When deploying manually using the %SYSTEM.Cluster API or the Management Portal, you can use the manual procedure described in Memory and Startup Settings.

  For guidelines for allocating memory to an InterSystems IRIS instance's routine and database caches as well as the shared memory heap, see Shared Memory Allocations .
- *Default globals database* indicates the target size of the database in question, which is the maximum expected size plus a margin for greater than expected growth. The file system hosting the database should be able to accommodate this total, with a safety margin there as well. For general information about InterSystems IRIS database size and expansion and the management of free space relative to InterSystems IRIS databases, and procedures for specifying database size and other characteristics when configuring instances manually, see Configuring Databases and Maintaining Local Databases.

  When deploying with the IKO, you can specify the size of the instance's storage volume for data, which is where the default globals databases for the master and cluster namespaces are located, as part of deployment; this must be large enough to accommodate the target size of the default globals database.

Important: When deploying manually, ensure that all instances have database directories and journal directories located on separate storage devices. This is particularly important when high volume data ingestion is concurrent with running queries. For guidelines for file system and storage configuration, including journal storage, see Storage Planning, File System Separation, and Journaling Best Practices.

- The number of data nodes (*NodeCount*) and the database cache size on each data node are both variables. The desired relationship between the sum of the data nodes' database cache sizes and the total working set estimates can be created by varying the number of shards and the database cache size per data node. This choice can be based on factors such as the tradeoff between system costs and memory costs; where more systems with lower memory resources are available, you can allocate smaller amounts of memory to the database caches, and when memory per system is higher, you can configure fewer servers. Generally speaking and all else being equal, more shards will perform faster due to the added parallelism, up to a point of diminishing returns (caused in part by increased sharding manager overhead). The most favorable configuration is typically in the 4-16 shard range, so other factors aside, for example, 8 data nodes with *M* memory each are likely to perform better than 64 shards with *M*/8 memory each.
- Bear in mind that if you need to add data nodes after the cluster has been loaded with data, you can automatically redistribute the sharded data across the new servers, which optimizes performance; see Add Data Nodes and Rebalance Data for more information. On the other hand, you cannot remove a data node with sharded data on it, and a server's sharded data cannot be automatically redistributed to other data nodes, so adding data nodes to a production cluster involves considerably less effort than reducing the number of data nodes, which requires dropping all sharded tables before removing the data nodes, then reloading the data after.
- Parallel query processing is only as fast as the slowest data node, so the best practice is for all data nodes in a sharded cluster to have identical or at least closely comparable specifications and resources. In addition, the configuration of all InterSystems IRIS instances in the cluster should be consistent; database settings such as collation and those SQL settings configured at instance level (default date format, for example) should be the same on all nodes to ensure correct SQL query results. Standardized procedures and use of an automated deployment method can help ensure this consistency.

The recommendations in the following table assume that you have followed the procedures for estimating total data and working set sizes described in the foregoing, including adding a 50% safety factor to the results of your calculations for each variable.

Table 3–2: Cluster Planning Guidelines

Size of ...	should be at least ...	Notes
Database cache on data nodes	(*ShardSizeWS* / *NodeCount*) + *NonshardSizeJoinedWS*	This recommendation assumes that your application requires 100% in-memory caching. Depending on the extent to which reads can be made from fast storage such as solid-state drives instead, the size of the cache can be reduced.
Default globals database for cluster namespace on each data node	*ShardSize* / *NodeCount* plus space for expected growth	When data ingestion performance is a major consideration, consider configuring initial size of the database to equal the expected maximum size, thereby avoiding the performance impact of automatic database expansion. However, if running in a cloud environment, you should also consider the cost impact of paying for storage you are not using.
Default globals database for master namespace on node 1 (see Configuring Namespaces)	*NonshardSizeTotal* and possibly space for expected growth	Nonsharded data is likely to grow less over time than sharded data, but of course this depends on your application.
IRISTEMP database on each data node (temporary storage database for master and cluster namespaces)	No specific guideline. The ideal initial size depends on your data set, workload, and query syntax, but will probably be in excess of 100 GB and could be considerably more.	Ensure that the database is located on the fastest possible storage, with plenty of space for significant expansion. T
CPU	No specific recommendations.	All InterSystems IRIS servers can benefit by greater numbers of CPUs, whether or not sharding is involved. Vertical scaling of CPU, memory, and storage resources can always be used in conjunction with sharding to provide additional benefit, but is not specifically required, and is governed by the usual cost/performance tradeoffs.

Important: All InterSystems IRIS instances in a sharded cluster must be of the same version, and all must have sharding licenses.

All data nodes in a sharded cluster should have identical or at least closely comparable specifications and resources; parallel query processing is only as fast as the slowest data node. In addition, the configuration of all InterSystems IRIS instances in the cluster should be consistent; database settings such as collation and those SQL settings configured at instance level (default date format, for example) should be the same on all nodes to ensure correct SQL query results. Standardized procedures and use of an automated deployment method can help ensure this consistency.

Because applications can connect to any data node's cluster namespace and experience the full dataset as if it were local, the general recommended best practice is to load balance application connections across all of the data nodes in a cluster. The IKO can automatically provision and configure a load balancer for the data nodes as needed under typical scenarios; if deploying a sharded cluster by other means, a load balancing mechanism is required. For an important discussion of load balancing a web server tier distributing application connections across data nodes, see Load Balancing, Failover, and Mirrored Configurations.

To maximize the performance of the cluster, it is a best practice to configure low-latency network connections between all of the data nodes, for example by locating them on the same subnet in the same data center or availability zone.

3.6.1.3 Plan Compute Nodes

As described in Overview of InterSystems IRIS Sharding, compute nodes cache the data stored on data nodes and automatically process read-only queries, while all write operations (insert, update, delete, and DDL operations) are executed on the data nodes. The scenarios most likely to benefit from the addition of compute nodes to a cluster are as follows:

- When high volume data ingestion is concurrent with high query volume, one compute node per data node can improve performance by separating the query workload (compute nodes) from the data ingestion workload (data nodes)
- When high multiuser query volume is a significant performance factor, multiple compute nodes per data node increases overall query throughput (and thus performance) by permitting multiple concurrent sharded queries to run against the data on each underlying data node. (Multiple compute nodes do not increase the performance of individual sharded queries running one at a time, which is why they are not beneficial unless multiuser query workloads are involved.) Multiple compute nodes also maintain workload separation when a compute node fails, as queries can still be processed on the remaining compute nodes assigned to that data node.

When planning compute nodes, consider the following factors:

- If you are considering deploying compute nodes, the best approach is typically to evaluate the operation of your basic sharded cluster before deciding whether the cluster can benefit from their addition. Compute nodes can be easily added to an existing cluster using one of the automated deployment methods described in Automated Deployment Methods for Clusters or using the %SYSTEM.Cluster API. For information adding compute nodes, see Deploy Compute Nodes for Workload Separation and Increased Query Throughput.
- For best performance, a cluster's compute nodes should be colocated with the data nodes (that is, in the same data center or availability zone) to minimize network latency.
- When compute nodes are added to a cluster, they are automatically distributed as evenly as possible across the data nodes. Bear in mind that adding compute nodes yields significant performance improvement only when there is at least one compute node per data node.
- The recommended best practice is to assign the same number of compute nodes to each data node. Therefore, if you are planning eight data nodes, for example, recommended choices for the number of compute nodes include zero, eight, sixteen, and so on.

- Because compute nodes support query execution only and do not store any data, their hardware profile can be tailored to suit those needs, for example by emphasizing memory and CPU and keeping storage to the bare minimum. All compute nodes in a sharded cluster should have closely comparable specifications and resources.
- Follow the data node database cache size recommendations (see Plan a Basic Cluster of Data Nodes) for compute nodes; ideally, each compute node should have the same size database cache as the data node to which it is assigned.

The distinction between data and compute nodes is completely transparent to applications, which can connect to any node's cluster namespace and experience the full dataset as if it were local. Application connections can therefore be load balanced across all of the data and compute nodes in a cluster, and under most applications scenarios this is the most advantageous approach. What is actually best for a particular scenario depends on whether you would prefer to optimize query processing or data ingestion. If sharded queries are most important, you can prioritize them by load balancing across the data nodes, so applications are not competing with shard-local queries for compute node resources; if high-speed ingestion using parallel load is most important, load balance across the compute nodes to avoid application activity on the data nodes. If queries and data ingestion are equally important, or you cannot predict the mix, load balance across all nodes.

The IKO allows you to automatically add a load balancer to your DATA node or COMPUTE node definitions; you can also create your own load balancing arrangement. For an important discussion of load balancing a web server tier distributing application connections across data nodes, see Load Balancing, Failover, and Mirrored Configurations.

3.6.2 Coordinated Backup and Restore of Sharded Clusters

When data is distributed across multiple systems, as in an InterSystems IRIS sharded cluster, backup and restore procedures may involve additional complexity. Where strict consistency of the data across a sharded cluster is required, independently backing up and restoring individual nodes is insufficient, because the backups may not all be created at the same logical point in time. This makes it impossible to be certain, when the entire cluster is restored following a failure, that ordering is preserved and the logical integrity of the restored databases is thereby ensured.

For example, suppose update A of data on data node S1 was committed before update B of data on data node S2. Following a restore of the cluster from backup, logical integrity requires that if update B is visible, update A must be visible as well. But if backups of S1 and S2 are taken independently, it is impossible to guarantee that the backup of S1 was made after A was committed, even if the backup of S2 was made after B was committed, so restoring the backups independently could lead to S1 and S2 being inconsistent with each other.

If, on the other hand, the procedures used coordinate either backup or restore and can therefore guarantee that all systems are restored to the same logical point in time — in this case, following update B — ordering is preserved and the logical integrity that depends on it is ensured. This is the goal of coordinated backup and restore procedures.

To greatly reduce the chances of having to use any of the procedures described here to restore your sharded cluster, you can deploy it with mirrored data servers, as described in Mirror for High Availability. Even if the cluster is unmirrored, most data errors (data corruption, for example, or accidental deletion of data) can be remedied by restoring the data node on which the error occurred from the latest backup and then recovering it to the current logical point in time using its journal files. The procedures described here are for use in much rarer situations requiring a cluster-wide restore.

This section covers the following topics:

- Coordinated backup and restore approaches for sharded clusters
- Coordinated backup and restore API calls
- Procedures for coordinated backup and restore

3.6.2.1 Coordinated Backup and Restore Approaches for Sharded Clusters

Coordinated backup and restore of a sharded cluster always involves all of the data nodes in the cluster. The InterSystems IRIS Backup API includes a Backup.ShardedCluster class that supports three approaches to coordinated backup and restore of a cluster's data nodes.

Bear in mind that the goal of all approaches is to *restore* all data servers to the same logical point in time, but the means of doing so varies. In one, it is the backups themselves that share a logical point in time, but in the others, InterSystems IRIS database journaling provides the common logical point in time, called a *journal checkpoint*, to which the databases are restored. The approaches include:

- Coordinated backups
- Uncoordinated backups followed by coordinated journal checkpoints
- A coordinated journal checkpoint included in uncoordinated backups

To understand how these approaches work, it is important that you understand the basics of InterSystems IRIS data integrity and crash recovery, which are discussed in Introduction to Data Integrity. Database journaling, a critical feature of data integrity and recovery, is particularly significant for this topic. Journaling records all updates made to an instance's databases in journal files. This makes it possible to recover updates made between the time a backup was taken and the moment of failure, or another selected point, by restoring updates from the journal files following restore from backup. Journal files are also used to ensure transactional integrity by rolling back transactions that were left open by a failure. For detailed information about journaling, see Journaling.

Considerations when selecting an approach to coordinated backup and restore include the following:

- The degree to which activity is interrupted by the backup procedure.
- The frequency with which the backup procedure should be performed to guarantee sufficient recoverability.
- The complexity of the required restore procedure.

These issues are discussed in detail later in this section.

3.6.2.2 Coordinated Backup and Restore API Calls

The methods in the Backup.ShardedCluster class can be invoked on any data node. All of the methods take a *ShardMasterNamespace* argument; this is the name of the master namespace on data node 1 (**IRISDM** by default) that is accessible from all nodes in the cluster.

The available methods are as follows:

- **Backup.ShardedCluster.Quiesce()**

  Blocks all activity on all data nodes of the sharded cluster, and waits until all pending writes have been flushed to disk. Backups of the cluster's data nodes taken under **Quiesce()** represent a logical point in time.

- **Backup.ShardedCluster.Resume()**

  Resumes activity on the data nodes after they are paused by **Quiesce()**.

- **Backup.ShardedCluster.JournalCheckpoint()**

  Creates a coordinated journal checkpoint and switches each data node to a new journal file, then returns the checkpoint number and the names of the *precheckpoint* journal files. The precheckpoint files are the last journal files on each data node that should be included in a restore; later journal files contain data that occurred after the logical point in time represented by the checkpoint.

- **Backup.ShardedCluster.ExternalFreeze**

  Freezes physical database writes, but not application activity, across the cluster, and then creates a coordinated journal checkpoint and switches each data node to a new journal file, returning the checkpoint number and the names of the precheckpoint journal files. The backups taken under **ExternalFreeze()** do not represent a logical point in time, but they include the precheckpoint journal files, thus enabling restore to the logical point in time represented by the checkpoint.

- **Backup.ShardedCluster.ExternalThaw**

Resumes disk writes after they are suspended by **ExternalFreeze()**.

You can review the technical documentation of these calls in the InterSystems Class Reference.

3.6.2.3 Procedures for Coordinated Backup and Restore

The steps involved in the three coordinated backup and restore approaches provided by the Sharding API are described in the following sections.

- Create coordinated backups

  Quiesces all database activity for a period of time.

- Create uncoordinated backups followed by coordinated journal checkpoints

  Zero downtime required.

- Include a coordinated journal checkpoint in uncoordinated backups

  Zero downtime required.

Data node backups should, in general, include not only database files but all files used by InterSystems IRIS, including the journal directories, write image journal, and installation data directory, as well as any needed external files. The locations of these files depend in part on how the cluster was deployed (see Deploying the Sharded Cluster); the measures required to include them in backups may have an impact on your choice of approach.

Important: The restore procedures described here assume that the data node being restored has no mirror failover partner available, and would be used with a mirrored data node only in a disaster recovery situation, as described in Disaster Recovery of Mirrored Sharded Clusters, as well as in Disaster Recovery Procedures. If the data node is mirrored, remove the primary from the mirror, complete the restore procedure described, and then rebuild it as described in Rebuilding a Mirror Member.

Create Coordinated Backups

1. Call **Backup.ShardedCluster.Quiesce**, which pauses activity on all data nodes in the cluster (and thus all application activity) and waits until all pending writes have been flushed to disk. When this process is completed and the call returns, all databases and journal files across the cluster are at the same logical point in time.
2. Create backups of all data nodes in the cluster. Although the database backups are coordinated, they may include open transactions; when the data nodes are restarted after being restored from backup, InterSystems IRIS recovery uses the journal files to restore transactional integrity by rolling back these back.
3. When backups are complete, call **Backup.ShardedCluster.Resume** to restore normal data node operation.

   Important: **Resume()** must be called within the same job that called **Quiesce()**. A failure return may indicate that the backup images taken under **Quiesce()** were not reliable and may need to be discarded.

4. Following a failure, on each data node:
   a. Restore the backup image.
   b. Verify that the *only* journal files present are those in the restored image from the time of the backup.

      CAUTION: This is critically important because at startup, recovery restores the journal files and rolls back any transactions that were open at the time of the backup. If journal data later than the time of the backup exists at startup, it could be restored and cause the data node to be inconsistent with the others.

   c. Restart the data node.

The data node is restored to the logical point in time at which database activity was quiesced.

Note: As an alternative to the first three steps in this procedure, you can gracefully shut down all data nodes in the cluster, create cold backups, and restart the data nodes.

Create Uncoordinated Backups Followed by Coordinated Journal Checkpoints

1. Create backups of the databases on all data nodes in the cluster while the data nodes are in operation and application activity continues. These backups may be taken at entirely different times using any method of your choice and at any intervals you choose.
2. Call **Backup.ShardedCluster.JournalCheckpoint()** on a regular basis, preferably as a scheduled task. This method creates a coordinated journal checkpoint and returns the names of the last journal file to include in a restore on each data node in order to reach that checkpoint. Bear in mind that it is the time of the latest checkpoint and the availability of the precheckpoint journal files that dictate the logical point in time to which the data nodes can be recovered, rather than the timing of the backups.

   Note: Before switching journal files, **JournalCheckpoint()** briefly quiesces all data nodes in the sharded cluster to ensure that the precheckpoint files all end at the same logical moment in time; as a result, application activity may be very briefly paused during execution of this method.

3. Ensure that for each data node, you store a complete set of journal files from the time of its last backup to the time at which the most recent coordinated journal checkpoint was created, ending with the precheckpoint journal file, and that all of these files will remain available following a server failure (possibly by backing up the journal files regularly). The databases backups are not coordinated and may also include partial transactions, but when the data nodes are restarted after being restored from backup, recovery uses the coordinated journal files to bring all databases to the same logical point in time and to restore transactional integrity.
4. Following a failure, identify the latest checkpoint available as a common restore point for all data nodes. This requires means that for each data node you have a database backup that preceding the checkpoint and all intervening journal files up to the precheckpoint journal file.

   CAUTION: This is critically important because at startup, recovery restores the journal files and rolls back any transactions that were open at the time of the backup. If journal files later than the precheckpoint journal file exist at startup, they could be restored and cause the data node to be inconsistent with the others.

5. On each data node, restore the databases from the backup preceding the checkpoint, restoring journal files up to the checkpoint. Ensure that no journal data after that checkpoint is applied. The simplest way to ensure that is to check if the server has any later journal files, and if so move or delete them, and then delete the journal log.

   The data node is now restored to the logical point in time at which the coordinated journal checkpoint was created.

Include a Coordinated Journal Checkpoint in Uncoordinated Backups

1. Call **Backup.ShardedCluster.ExternalFreeze**(). This method freezes all activity on all data nodes in the sharded cluster by suspending their write daemons; application activity continues, but updates are written to the journal files only and are not committed to disk. Before returning, the method creates a coordinated journal checkpoint and switches each data node to a new journal file, then returns the checkpoint number and the names of the precheckpoint journal files. At this point, the precheckpoint journal files represent a single logical point in time.

Note: **Backup.ShardedCluster.ExternalFreeze**() internally calls **Backup.ShardedCluster.JournalCheckpoint()**, which in turn calls **Backup.ShardedCluster.Quiesce()** and **Backup.ShardedCluster.Resume()** to briefly quiesce the system while switching journal files. When **Resume()** completes it logs the message `Backup.ShardedCluster.Resume: System resumed`. This does not mean that the system is no longer frozen, only that it is no longer quiesced; after calling **ExternalFreeze()**, the system remains frozen until **Backup.ShardedCluster.ExternalThaw()** is called.

2. Create backups of all data nodes in the cluster. The databases backups are not coordinated and may also include partial transactions, but when restoring the data nodes you will ensure that they are recovered to the journal checkpoint, bringing all databases to the same logical point in time and to restoring transactional integrity.

   Note: By default, when the write daemons have been suspended by **Backup.ShardedCluster.ExternalFreeze()** for 10 minutes, application processes are blocked from making further updates (due to the risk that journal buffers may become full). However, this period can be extended using an optional argument to **ExternalFreeze()** if the backup process requires more time.

3. When all backups are complete, call **Backup.ShardedCluster.ExternalThaw()** to resume the write daemons and restore normal data node operation.

   Important: A failure return may indicate that the backup images taken under **ExternalFreeze()** were not reliable and may need to be discarded.

4. Following a failure, on each data node:

   a. Restore the backup image.

   b. Remove any journal files present in the restored image that are later than the precheckpoint journal file returned by **ExternalFreeze()**.

   c. Follow the instructions in Starting InterSystems IRIS Without Automatic WIJ and Journal Recovery to manually recover the InterSystems IRIS instance. When you restore the journal files, start with the journal file that was switched to by **ExternalFreeze()** and end with the precheckpoint journal file returned by **ExternalFreeze()**. (Note that these may be the same file, in which case this is the one and only journal file to restore.)

      Note: If you are working with containerized InterSystems IRIS instances, see Upgrading When Manual Startup is Required for instructions for doing a manual recovery inside a container.

   The data node is restored to the logical point in time at which the coordinated journal checkpoint was created by the **ExternalFreeze()** method.

Note: This approach requires that the databases and journal files on each data node be located such that a single backup can include them both.

3.6.3 Disaster Recovery of Mirrored Sharded Clusters

Disaster recovery (DR) asyncs keep the same synchronized copies of mirrored databases as the backup failover member, the differences being that communication between an async and its primary is asynchronous, and that an async does not participate in automatic failover. However, a DR async can be promoted to failover member, becoming the backup, when one of the failover members has become unavailable; for example, when you are performing maintenance on the backup, or when an outage of the primary causes the mirror to fail over to the backup and you need to maintain the automatic failover capability while you investigate and correct the problem with the former primary. When a major failure results in an outage of both failover members, you can perform disaster recovery by manually failing over to a promoted DR async.

DR async mirror members make it possible to provide a disaster recovery option for a mirrored sharded cluster, allowing you to restore the cluster to operation following an outage of the mirror failover pairs in a relatively short time. Specifically, enabling disaster recovery for a mirrored cluster includes:

- Configuring at least one DR async in every data node mirror in a mirrored sharded cluster.

  To help ensure that they remain available when a major failure creates a failover pair outage, DR asyncs are typically located in separate data centers or availability zones from the failover pairs. If the DR asyncs you manually fail over to in your disaster recovery procedure are distributed across multiple locations, the network latency between them may have a significant impact on the performance of the cluster. For this reason, it is a best practice to ensure that all of the data node mirrors in the cluster include at least one DR async in a common location.

  Important: If you add DR asyncs to the data node mirrors in an existing mirrored cluster as part of enabling disaster recovery (or for any other reason), or demote a backup member to DR async, you must call **$SYSTEM.Sharding.VerifyShards()** in the cluster namespace on one of the mirror primaries to update the cluster's metadata for the additions.

- Making regular coordinated backups as described in the previous section.

  Because the degree to which the cluster's operation is interrupted by backups, the frequency with which backups should be performed, and the complexity of the restore procedure all vary with the coordinated backup and restore approach you choose, you should review these approaches to determine which is most appropriate to your circumstances and disaster recovery goals (the amount of data loss you are willing to tolerate, the speed with which cluster operation must be restored, and so on).

- Planning, preparing, and testing the needed disaster recovery procedures, including the restore procedure described for the coordinated backup procedure you have selected, as well as familiarizing yourself with the procedures described in Disaster Recovery Procedures.

Assuming you have configured the needed DR asyncs and have been making regular coordinated backups, the general disaster recovery procedure for a mirrored sharded cluster would be as follows:

1. In each data node mirror, do the following:
   - Promote a DR async (ideally one sharing a common location with DR asyncs in all of the other data node mirrors) to failover member using the procedures described in Promoting a DR Async Member to Failover Member.
   - Manually fail over to the promoted DR async, making it primary, as described in Manual Failover to a Promoted DR Async During a Disaster.
2. Restore the most recent coordinated backup using the procedures described for the coordinated backup and restore approach you selected, as described in the appropriate section of Procedures for Coordinated Backup and Restore.
3. To restore failover capability to the cluster, complete the failover pair in each mirror. If the data node mirrors all included multiple DR asyncs, promote another DR async to failover member in each. If there are no additional DR asyncs in the mirrors, configure a second failover member for each as described in Mirror Data Nodes for High Availability.
4. Restore application access to the sharded cluster.

Note: If the mirrored sharded cluster you recovered included compute nodes, these were very likely colocated with the data node failover pairs, and also unavailable due to the failure. In this case, a full recovery of the cluster would include minimizing network latency by deploying new compute nodes colocated with the recovered cluster, as described in Deploy Compute Nodes for Workload Separation and Increased Query Throughput. If the cluster's existing compute nodes are still operational in the original location, they should be relocated to the new cluster location as soon as possible. A recovered cluster is operational without the compute nodes, but is lacking the benefits they provided.

3.6.4 Cloning a Sharded Cluster

Under some circumstances you may need to replicate an existing cluster on a different set of hosts, for example when you want to:

- Stand up a test system based on a snapshot of a production cluster.
- Restore a cluster from backup on new hosts following a failure that rendered some or all of the existing hosts unusable.
- Move a cluster to a new location, such as from one data center to another.

This section describes a procedure for replicating an existing cluster on a new set of hosts, so that you can accomplish one of these tasks or something similar.

A sharded cluster is made up of nodes and namespaces that are configured to work together through multiple mappings and communication channels. Information about this configuration, including specific hostnames, port numbers, and namespaces, is stored and maintained in the master namespace on data node 1 (or the shard master in a namespace-level cluster). To replicate a cluster, you need to duplicate this configuration on the new cluster.

Note: This procedure can be used only when the number of data nodes in the original (existing) and target (new) clusters are the same.

Because %SYSTEM.Sharding API calls are involved in the cloning process, the resulting cloned cluster is always a namespace-level cluster, which can be managed and modified using only the %SYSTEM.Sharding API and the namespace-level pages in the Management Portal.

The instructions for the procedure assume one node per host, but can be adapted for use with a namespace-level cluster with multiple nodes per host.

To clone a sharded cluster, follow these steps:

1. Provision or identify the data node hosts for the target cluster, deploy InterSystems IRIS on them, and enable sharding using the Management Portal or the $SYSTEM.Sharding.EnableSharding API call.
2. In the master namespace on data node 1 of the original cluster, call **$SYSTEM.Sharding.GetConfig()** to display the locations of the default globals database of the master namespace on node 1 and the default globals database of the shard namespace on each data node. These databases must be replicated on the corresponding nodes in the new cluster, so be sure to record the information accurately. If the default routines database of the master namespace is different from its globals database, include it in the information you record.

   The output of GetConfig() includes the shard number of each data node; you will need each original data node's shard number to configure the corresponding target data node. Determine which target node will replicate which original node by assigning one of the original cluster's shard numbers to each host in the target cluster.
3. Block application access to the original cluster. When all writes have been flushed to disk, individually shut down all of the InterSystems IRIS instances in the cluster.
4. Replicate the original cluster's globals databases (and possibly the master's routines database) on the target cluster. You can do this in one of two ways:
   - Back up the databases on the original cluster and restore them on the target cluster.
   - Copy the IRIS.DAT files from the original cluster to the target cluster.

   The directory paths do not need to be the same on the two clusters. Whichever method you use, ensure that you replicate each database on the target cluster node corresponding to the original's shard number.

   Note: If the original cluster is mirrored, back up or copy the databases from just one failover member of each node.

5. On each node of the target cluster, create a shard namespace with the replicated shard database as its default globals database. On node 1, do the same for the master namespace; include the routines database if it is different from the globals database.
6. In the master namespace on node 1 of the target cluster:
   a. Call **$SYSTEM.Sharding.ReassignShard()** once for each target data node, specifying the hostname or IP address, superserver port, shard namespace, and shard number of each, making sure to specify the correct shard numbers to correspond to the original cluster, for example:

      ```
      set status =
      $SYSTEM.Sharding.ReassignShard(,"shard-host",shard-port,"shard-namespace","shard-number")
      ```

   b. Call **$SYSTEM.Sharding.Reinitialize()**, which verifies that the target cluster is of a compatible version; sets up all the mappings, ECP connections, and metadata needed to activate the target cluster; and automatically calls **$SYSTEM.Sharding.VerifyShards()** to complete activation and confirm that the configuration is correct.

      If the master namespace contains any user-defined global, routine, or package mappings to databases other than the globals and routines databases you replicated on the target cluster, Reinitialize() returns an error because these additional databases are not available in the new cluster. You can avoid the error by specifing `1` for the second, `IgnoreMappings` argument when you call Reinitialize(), as follows:

      ```
      set status = $SYSTEM.Sharding.Reinitialize(,1)
      ```

      Following the Reinitialize() call, replicate the additional databases involved on target node 1, as you did the globals and routines databases, and when they are all in place call **$SYSTEM.Sharding.VerifyShards()** to propagate the relevant mappings to the shard namespaces.
7. If desired, convert the cloned cluster to mirrored, add compute nodes, or both.

3.6.5 Sharding APIs

At this release, InterSystems IRIS provides two APIs for use in configuring and managing a sharded cluster:

- The %SYSTEM.Cluster API is for use in deploying and managing the current architecture (see Elements of Sharding).
- The %SYSTEM.Sharding API is for use in deploying and managing the namespace-level architecture of previous versions (see Namespace-level Sharding Architecture).

3.6.5.1 %SYSTEM.Cluster API

For more detail on the %SYSTEM.Cluster API methods and instructions for calling them, see the %SYSTEM.Cluster class documentation in the InterSystems Class Reference.

Use the %SYSTEM.Cluster API methods in the following ways:

- Set up an InterSystems IRIS instance as the first node of a new sharded cluster by calling **Initialize**.
- Add an instance to a cluster as a data node by calling **AttachAsDataNode on the instance being added**.
- Add an instance to a cluster as a compute node by calling **AttachAsComputeNode** on the instance being added.
- Display a list of a cluster's nodes by calling **ListNodes**.
- Retrieve an overview of a cluster's metadata by calling **GetMetaData**.
- Get the name of the cluster namespace for the current instance by calling **ClusterNamespace**.

%SYSTEM.Cluster methods include the following; click the name of a method to open its Class Reference entry.

- **$SYSTEM.Cluster.Initialize()**

  Automatically and transparently performs all steps needed to enable the current InterSystems IRIS instance as the first node of a cluster

- **$SYSTEM.Cluster.AttachAsDataNode()**

  Attaches the current InterSystems IRIS instance to a specified cluster as a data node.

- **$SYSTEM.Cluster.AttachAsComputeNode()**

  Attaches the current InterSystems IRIS instance to a specified cluster as a compute node.

- **$SYSTEM.Cluster.ListNodes()**

  Lists the nodes of the cluster to which the current InterSystems IRIS instance belongs to the console or to a specified output file.

- **$SYSTEM.Cluster.GetMetaData()**

  Retrieves an overview of the metadata of the cluster to which the current InterSystems IRIS instance belongs.

- **$SYSTEM.Cluster.ClusterNamespace()**

  Gets the name of the cluster namespace on the current InterSystems IRIS instance.

3.6.5.2 %SYSTEM.Sharding API

For more detail on the %SYSTEM.Sharding API methods and instructions for calling them, see the %SYSTEM.Sharding class documentation in the InterSystems Class Reference.

Use the %SYSTEM.Sharding API methods in the following ways:

- Enable an InterSystems IRIS instance to act as a shard master or shard server by calling the **EnableSharding** method.
- Define the set of shards belonging to a master namespace by making repeated calls to **AssignShard** in the master namespace, one call for each shard.
- Once shards have been assigned, verify that they are reachable and correctly configured by calling **VerifyShards**.
- If additional shards are assigned to a namespace that already contains sharded tables, and the new shards can't be reached for automatic verification during the calls to **AssignShard**, you can call **ActivateNewShards** to activate them once they are reachable.
- List all the shards assigned to a master namespace by calling **ListShards**.
- When converting a nonmirrored cluster to a mirrored cluster, after creating a mirror on each existing data node, add the master database and shard databases to their respective mirrors by calling AddDatabasesToMirrors.
- Rebalance existing sharded data across the cluster after adding data nodes/shard data servers with **$SYSTEM.Sharding.Rebalance()** (see Add Data Nodes and Rebalance Data).
- Assign a shard data server to a different shard namespace at a different address by calling **ReassignShard**.
- Remove a shard from the set belonging to a master namespace by calling **DeassignShard**.
- Set sharding configuration options by calling **SetOption**, and retrieve their current values by calling **GetOption**.

%SYSTEM.Sharding methods include the following; click the name of a method to open its Class Reference entry.

- **$SYSTEM.Sharding.EnableSharding()**

  Enables the current InterSystems IRIS instance to act as a shard master or shard server.

- **$SYSTEM.Sharding.AssignShard()**

Assigns a shard to a master namespace.

- **$SYSTEM.Sharding.VerifyShards()**

  Verifies that assigned shards are reachable and are correctly configured.

- **$SYSTEM.Sharding.ListShards()**

  Lists the shards assigned to a specified master namespace, to the console or current device.

- **$SYSTEM.Sharding.ActivateNewShards()**

  Activates shards that could not be activated by prior calls to **AssignShard**.

- **$SYSTEM.Sharding.AddDatabasesToMirrors()**

  Adds the master and shard databases of data nodes that have been added as mirror failover members to their respective mirrors.

- **$SYSTEM.Sharding.Rebalance()**

  Rebalances existing sharded data across the cluster after adding data nodes/shard data servers.

- **$SYSTEM.Sharding.ReassignShard()**

  Reassigns a shard by assigning its shard number to a different shard namespace at a different address.

- **$SYSTEM.Sharding.DeassignShard()**

  Deassigns (unassigns) a shard from a master namespace to which it had previously been assigned. This removes the shard from the set of shards belonging to the master namespace.

- **$SYSTEM.Sharding.SetOption()**

  Sets a specified sharding configuration option to a specified value within the scope of a specified master namespace.

- **$SYSTEM.Sharding.GetOption()**

  Gets the value of a specified sharding configuration option within the scope of a specified master namespace.

- **$SYSTEM.Sharding.SetNodeIPAddress()**

  Configures a specified IP address rather than a node's hostname as its address for cluster communications (must be used on all nodes).

- **$SYSTEM.Sharding.GetConfig()**

  Retrieves configuration information about the sharded cluster to which the specified master or cluster namespace belongs.

- **$SYSTEM.Sharding.Reinitialize()**

  Reinitializes the internal mappings, connections, and cluster metadata of a sharded cluster which has been cloned (as described in Cloning a Sharded Cluster).

- **$SYSTEM.Sharding.GetFederatedTableInfo()**

  Returns information about the specified federated table.

- **$SYSTEM.Sharding.Help()**

  Displays a summary of the methods of %SYSTEM.Sharding.

3.6.6 Deploying the Namespace-level Architecture

Use the procedures in this section to deploy an InterSystems IRIS sharded cluster with the older namespace-level architecture, consisting of a shard master, shard data servers, and optionally query servers using the Management Portal or the %SYSTEM.Sharding. These procedures assume each InterSystems IRIS instance is installed on its own system.

Because applications can connect to any shard data server's cluster namespace and experience the full dataset as if it were local, the general recommended best practice for a namespace-level cluster that does not include shard query servers is to load balance application connections across all of the shard data servers. In general, this is also the best approach for clusters that include shard query servers, but there are additional considerations; for more information, see the end of Plan Compute Nodes.

Unlike node-level deployment of a node-level cluster, namespace-level deployment cannot create mirrors as part of cluster configuration. However, if you want to deploy a mirrored sharded cluster — that is, with a mirrored shard master data server and mirrored shard data servers — you can do any of the following:

- Configure the primary of an existing mirror as the shard master data server, first adding the globals database of the intended master namespace to the mirror.
- Create a mirror with the existing shard master data server as primary, adding the globals database of the master namespace to the mirror before you add the backup.
- Assign the failover members of an existing mirror as a shard data server, first adding the globals database of the intended cluster namespace to the mirror.
- Create a mirror with an assigned shard data server as primary, adding the globals database of the shard namespace to the mirror, and then provide information about the backup to reassign it as a mirrored shard data server.

You can include DR asyncs in mirrored sharded clusters, but not reporting asyncs. Bear in mind that the recommended best practice is to avoid mixing mirrored and nonmirrored nodes, that is, the shard master and all shard data servers should be mirrored, or none of them should be. The steps of these procedures, shown below, can be used to deploy both nonmirrored and mirrored clusters.

- Provision or identify the infrastructure
- Deploy InterSystems IRIS on the cluster nodes
- Prepare the shard data servers
- Configure the cluster using either
  - the Management Portal
  - the %SYSTEM.Sharding API

3.6.6.1 Provision or Identify the Infrastructure

Identify the needed number of networked host systems (physical, virtual, or cloud) — one host each for the shard master and the shard data servers.

If you want to deploy mirror cluster, provision two hosts for the shard master data server, and two or more (depending on whether you want to a DR async members) for each shard data server.

Important: Be sure to review provision or identify the infrastructure for requirements and best practices for the infrastructure of a sharded cluster.

3.6.6.2 Deploy InterSystems IRIS on the Cluster Nodes

This procedure assumes that each system hosts or will host a single InterSystems IRIS instance.

1. Deploy an instance of InterSystems IRIS, either by creating a container from an InterSystems-provided image (as described in Running InterSystems Products in Containers) or by installing InterSystems IRIS from a kit (as described in the Installation Guide).

   Important: Be sure to review deploy InterSystems IRIS on the data nodes for requirements and best practices for the InterSystems IRIS instances in a sharded cluster.

2. Ensure that the storage device hosting each instance's databases is large enough to accommodate the target globals database size, as described in Estimate the Database Cache and Database Sizes.

   All instances should have database directories and journal directories located on separate storage devices, if possible. This is particularly important when high volume data ingestion is concurrent with running queries. For guidelines for file system and storage configuration, including journal storage, see Storage Planning, File System Separation, and Journaling Best Practices.

3. Allocate the database cache (global buffer pool) for each instance, depending on its eventual role in the cluster, according to the sizes you determined in Estimate the Database Cache and Database Sizes. For the procedure for allocating the database cache, see Memory and Startup Settings.

   Note: In some cases, it may be advisable to increase the size of the shared memory heap on the cluster members. For information on how to allocate memory to the shared memory heap, see gmheap.

   For guidelines for allocating memory to an InterSystems IRIS instance's routine and database caches as well as the shared memory heap, see System Resource Planning and Management.

3.6.6.3 Configure the Namespace-Level Cluster using the Management Portal

Once you have provisioned or identified the infrastructure and installed InterSystems IRIS on the cluster nodes, follow the steps in this section to configure the namespace-level cluster using the Management Portal.

For information about opening the Management Portal in your browser, see the instructions for an instance deployed in a container or one installed from a kit.

Prepare the Shard Data Servers

Do the following on each desired instance to prepare it to be added to the cluster as a shard data server.

Note: Under some circumstances, the API (which underlies the Management Portal) may be unable to resolve the hostnames of one or more nodes into IP addresses that are usable for interconnecting the nodes of a cluster. When this is the case, you can call **$SYSTEM.Sharding.SetNodeIPAddress()** (see %SYSTEM.Sharding API) to specify the IP address to be used for each node. To use $SYSTEM.Sharding.SetNodeIPAddress(), you *must* call it on every intended cluster node *before* making any other %SYSTEM.Sharding API calls on those nodes, for example:

```
set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
```

When this call is used, you must use the IP address you specify for each shard data server node, rather than the hostname, when assigning the shards to the master namespace, as described in Configure the Shard Master Data Server and Assign the Shard Data Servers.

1. Open the Management Portal for the instance and select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK**. (The value of the **Maximum Number of ECP Connections** setting need not be changed as the default is appropriate for virtually all clusters.)

2. Restart the instance. (There is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted.)

3. To create the shard namespace, navigate to the Configure Namespace-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Namespace-Level**) and click the **Create Namespace** button, then follow the instructions in Create/Modify a Namespace. (The namespace need not be interoperability-enabled.) Although the shard namespaces on the different shard data servers can have different names, you may find it more convenient to give them the same name.

   In the process, create a new database for the default globals database, making sure that it is located on a device with sufficient free space to accommodate the target size for the shard namespaces, as described in Estimate the Database Cache and Database Sizes. If data ingestion performance is a significant consideration, set the initial size of the database to its target size. Also select the globals database you created as the namespace's default routines database.

   Important: If you are preparing an existing mirror to be a shard data server, when you create the default globals database for the shard namespace on each member — the primary first, then the backup, then any DR asyncs — select **Yes** at the **Mirrored database?** prompt to include the shard namespace globals database in the mirror.

   Note: As noted in the Estimate the Database Cache and Database Sizes, the shard master data server and shard data servers all share a single default globals database physically located on the shard master and known as the *master globals database*. The default globals database created when a shard namespace is created remains on the shard, however, becoming the local globals database, which contains the data stored on the shard. Because the shard data server does not start using the master globals database until assigned to the cluster, for clarity, the planning guidelines and instructions in this document refer to the eventual local globals database as the default globals database of the shard namespace.

   A new namespace is automatically created with **IRISTEMP** configured as the temporary storage database; do not change this setting for the shard namespace.

4. For a later step, record the DNS name or IP address of the host system, the superserver (TCP) port of the instance, and the name of the shard namespace you created.

   Note: From the perspective of another node (which is what you need in this procedure), the superserver port of a containerized InterSystems IRIS instance depends on which host port the superserver port was published or exposed as when the container was created. For details on and examples of this, see Running an InterSystems IRIS Container with Durable %SYS and Running an InterSystems IRIS Container: Docker Compose Example and see Container networking in the Docker documentation.

   The default superserver port number of a noncontainerized InterSystems IRIS instance that is the only such on its host is 1972. To see or set the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal. (For information about opening the Management Portal for the instance and determining the superserver port, see the instructions for an instance deployed in a container or one installed from a kit.)

Configure the Shard Master Data Server and Assign the Shard Data Servers

Do the following on the desired instance to configure it as the shard master data server and assign the prepared shard data servers to the cluster. You can also use the procedure to assign query shards to the cluster.

1. Open the Management Portal for the instance, select **System Administration > Configuration > System Configuration > Sharding > Enable Sharding**, and on the dialog that displays, click **OK**. (The value of the **Maximum Number of ECP Connections** setting need not be changed as the default is appropriate for virtually all clusters.)

2. Restart the instance. (There is no need to close the browser window or tab containing the Management Portal; you can simply reload it after the instance has fully restarted.)

3. To create the master namespace, navigate to the Configure Namespace-Level page (**System Administration > Configuration > System Configuration > Sharding > Configure Namespace-Level**) and click the **Create Namespace** button, then follow the instructions in Create/Modify a Namespace. (The namespace need not be interoperability-enabled.)

   In the process, create a new database for the default globals database, making sure that it is located on a device with sufficient free space to accommodate the target size for the master namespace, as described in Estimate the Database Cache and Database Sizes. Also select the globals database you created as the namespace's default routines database.

   Important: If you are configuring an existing mirror as the shard master data server, when you create the default globals database for the master namespace on each member — the primary first, then the backup, then any DR asyncs — select **Yes** at the **Mirrored database?** prompt to include the master namespace globals database in the mirror.

4. To assign the shard data servers, ensure that the **Namespace** drop-down is set to the master namespace you just created. For each shard, press the **Assign Shard** button on the Configure Namespace-Level page to display the ASSIGN SHARD dialogue, then enter the following information for the shard data server instance:

   - The name of its host (or the IP address, if you used the **$SYSTEM.Sharding.SetNodeIPAddress()** call on each node before starting this procedure, as described at the beginning of this section).
   - The superserver port.
   - The name of the shard namespace you created when preparing it.
   - **Data** as the role for the shard.

     Important: To assign a shard query server, select **Query** as the role for the shard, rather than **Data**.

   If you are assigning an existing mirror as a shard data server, begin with the primary and, after entering the above information, select **Mirrored** and provide the following information; the backup is then automatically assigned with the primary. (Shard query servers are never mirrored.)

   - The name of the mirror.
   - The name (or IP address) of the backup's host.
   - The superserver port of the backup.
   - The mirror's virtual IP address (VIP), if it has one.

   Finally, select **Finish**.

5. After you assign a shard, it is displayed in the list on the Configure Namespace-Level page; if it is a mirror, both failover members are included.

6. After you have assigned all the shard data servers (and optionally shard query servers), click **Verify Shards** to confirm that the ports are correct and all needed configuration of the nodes is in place so that the shard master can communicate with the shard data servers.

Deassign a Shard Server

You can use the **Deassign** link to the right of a listed shard data server or shard query server to remove the shard server from the cluster. When deassigning a mirrored shard data server, you must do this in the Management Portal of the primary.

Important: If there are sharded tables or classes on the cluster (regardless of whether the tables contain data) you cannot deassign any of the shard data servers.

3.6.6.4 Configure the Cluster Nodes Using the API

Once you have provisioned or identified the infrastructure and installed InterSystems IRIS on the cluster nodes, follow the steps in this section to configure the namespace-level cluster using the API. Consult the %SYSTEM.Sharding class documentation in the InterSystems Class Reference for complete information about the calls described here, as well as other calls in the API.

Note: Under some circumstances, the API may be unable to resolve the hostnames of one or more nodes into IP addresses that are usable for interconnecting the nodes of a cluster. When this is the case, you can call **$SYSTEM.Sharding.SetNodeIPAddress()** (see %SYSTEM.Sharding API) to specify the IP address to be used for each node. To use $SYSTEM.Sharding.SetNodeIPAddress(), you *must* call it on every intended cluster node *before* making any other %SYSTEM.Sharding API calls on those nodes, for example:

```
set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
```

When this call is used, you must use the IP address you specify for each node, rather than the hostname, as the *shard-host* argument when calling **$SYSTEM.Sharding.AssignShard()** on the shard master to assign the node to the cluster, as described in the following procedure.

Prepare the Shard Data Servers

Do the following on each desired instance to prepare it to be added to the cluster as a shard data server.

Note: Namespace-level deployment cannot create mirrors. You can, however, add the failover members of existing mirrors as shard data servers. To do this, create the mirrors, prepare the members using this procedure, and then assign them as shard data servers as described in the next procedure.

1. Create the shard namespace using the Management Portal, as described in Create/Modify a Namespace. (The namespace need not be interoperability-enabled.) Although the shard namespaces on the different shard data servers can have different names, you ay find it more convenient to give them the same name.

   Create a new database as the default globals database, making sure that it is located on a device with sufficient free space to accommodate its target size, as described in Estimate the Database Cache and Database Sizes. If data ingestion performance is a significant consideration, set the initial size of the database to its target size.

   Important: If you are preparing an existing mirror to be a shard data server, when you create the default globals database for the shard namespace on each member — the primary first, then the backup, then any DR asyncs — select **Yes** at the **Mirrored database?** prompt to include the shard namespace globals database in the mirror.

   Select the globals database you created for the namespace's default routines database.

   Note: As noted in the Estimate the Database Cache and Database Sizes, the shard master data server and shard data servers all share a single default globals database physically located on the shard master and known as the *master globals database*. The default globals database created when a shard namespace is created remains on the shard, however, becoming the local globals database, which contains the data stored on the shard. Because the shard data server does not start using the master globals database until assigned to the cluster, for clarity, the planning guidelines and instructions in this document refer to the eventual local globals database as the default globals database of the shard namespace.

   A new namespace is automatically created with **IRISTEMP** configured as the temporary storage database; do not change this setting for the shard namespace.

2. For a later step, record the DNS name or IP address of the host system, the superserver (TCP) port of the instance, and the name of the shard namespace you created.

Note: From the perspective of another node (which is what you need in this procedure), the superserver port of a containerized InterSystems IRIS instance depends on which host port the superserver port was published or exposed as when the container was created. For details on and examples of this, see Running an InterSystems IRIS Container with Durable %SYS and Running an InterSystems IRIS Container: Docker Compose Example and see Container networking in the Docker documentation.

The default superserver port number of a noncontainerized InterSystems IRIS instance that is the only such on its host is 1972. To see or set the instance's superserver port number, select **System Administration** > **Configuration** > **System Configuration** > **Memory and Startup** in the instance's Management Portal. (For information about opening the Management Portal for the instance and determining the superserver port, see the instructions for an instance deployed in a container or one installed from a kit.)

3. In an InterSystems Terminal window, in any namespace, call **$SYSTEM.Sharding.EnableSharding** (see %SYSTEM.Sharding API) to enable the instance to participate in a sharded cluster, as follows:

```
set status = $SYSTEM.Sharding.EnableSharding()
```

No arguments are required.

Note: To see the return value (for example, 1 for success) for the each API call detailed in these instructions, enter:

```
zw status
```

Reviewing **status** after each call is a good general practice, as a call might fail silently under some circumstances. If a call does not succeed (**status** is not **1**), display the user-friendly error message by entering:

```
do $SYSTEM.Status.DisplayError(status)
```

After making this call, restart the instance.

Configure the Shard Master Data Server and Assign the Shard Data Servers

On the shard master data server instance, do the following to configure the shard master and assign the shard data servers.

1. Create the master namespace using the Management Portal, as described in Create/Modify a Namespace. (The namespace need not be interoperability-enabled.)

   Ensure that the default globals database you create is located on a device with sufficient free space to accommodate its target size, as described in Estimate the Database Cache and Database Sizes. If data ingestion performance is a significant consideration, set the initial size of the database to its target size.

   Important: If you are configuring an existing mirror as the shard master data server, when you create the default globals database for the master namespace on each member — the primary first, then the backup, then any DR asyncs — select **Yes** at the **Mirrored database?** prompt to include the master namespace globals database in the mirror.

   Select the globals database you created for the namespace's default routines database.

   Note: A new namespace is automatically created with **IRISTEMP** configured as the temporary storage database; do not change this setting for the master namespace. Because the intermediate results of sharded queries are stored in IRISTEMP, this database should be located on the fastest available storage with significant free space for expansion, particularly if you anticipate many concurrent sharded queries with large result sets.

2. In an InterSystems Terminal window, in any namespace, do the following:

a. Call **$SYSTEM.Sharding.EnableSharding()** (see %SYSTEM.Sharding API) to enable the instance to participate in a sharded cluster (no arguments are required), as follows:

```
set status = $SYSTEM.Sharding.EnableSharding()
```

After making this call, restart the instance.

b. Call **$SYSTEM.Sharding.AssignShard()** (see %SYSTEM.Sharding API) once for each shard data server, to assign the shard to the master namespace you created, as follows:

```
set status = $SYSTEM.Sharding.AssignShard("master-namespace","shard-host",shard-superserver-port,
    "shard_namespace")
```

where the arguments represent the name of the master namespace you created and the information you recorded for that shard data server in the previous step, for example:

```
set status = $SYSTEM.Sharding.AssignShard("master","shardserver3",1972,"shard3")
```

If you used the **$SYSTEM.Sharding.SetNodeIPAddress()** call on each node before starting this procedure, as described at the beginning of this section, substitute the shard host's IP address for *shard-host*, rather than its hostname.

c. To verify that you have assigned the shards correctly, you can issue the following command and verify the hosts, ports, and namespace names:

```
do $SYSTEM.Sharding.ListShards()
Shard   Host                    Port    Namespc  Mirror  Role    VIP
1       shard1.internal.acme.com   56775   SHARD1
2       shard2.internal.acme.com   56777   SHARD2
...
```

Note: For important information about determining the superserver port of an InterSystems IRIS instance, see step 2 in Configure the Shard Data Servers.

d. To confirm that the ports are correct and all needed configuration of the nodes is in place so that the shard master can communicate with the shard data servers, call **$SYSTEM.Sharding.VerifyShards()** (see %SYSTEM.Sharding API) as follows:

```
do $SYSTEM.Sharding.VerifyShards()
```

The **$SYSTEM.Sharding.VerifyShards()** call identifies a number of errors. For example, if the port provided in a **$SYSTEM.Sharding.AssignShard()** call is a port that is open on the shard data server host but not the superserver port for an InterSystems IRIS instance, the shard is not correctly assigned; the **$SYSTEM.Sharding.VerifyShards()** call indicates this.

e. If you prepared and added the failover pairs of mirrors as shard data servers, you can use the **$SYSTEM.Sharding.AddDatabasesToMirrors()** call (see %SYSTEM.Sharding API) on the shard master data server to add the master and shard databases as mirrored databases on each primary, fully configuring each failover pair as a mirrored shard data server, as follows:

```
set status = $SYSTEM.Sharding.AddDatabasesToMirrors("master-namespace")
```

Bear in mind that the recommended best practice is to avoid mixing mirrored and nonmirrored shard data servers, but rather to deploy the cluster as either entirely mirrored or entirely nonmirrored.

To add a DR async to the failover pair in a configured data node mirror, create databases on the new node corresponding to the mirrored databases on the first failover member, add the new node to the mirror as a DR async, and restore the databases from the backup made on the first failover member to automatically add them to the mirror.

Create a mirror on each existing data node and then call **$SYSTEM.Sharding.AddDatabasesToMirrors()** on node 1 to automatically convert the cluster to a mirrored configuration.

3.6.7 Reserved Names

The following names are used by InterSystems IRIS and should not be used in the names of user-defined elements:

- The package name **IRIS.Shard** is reserved for system-generated shard-local class names and should not be used for user-defined classes.
- The schema name **IRIS_Shard** is reserved for system-generated shard-local table names and should not be used for user-defined tables.
- The prefixes **IRIS.Shard.**, **IS.**, and **BfVY.** are reserved for globals of shard-local tables, and in shard namespaces are mapped to the shard's local databases. User-defined global names and global names for nonsharded tables should not begin with these prefixes. Using these prefixes for globals other than those of shard-local tables can result in unpredictable behavior.