Scalability Guide

Version 2020.3
2021-02-04

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

Scalability Guide
InterSystems IRIS Data Platform Version 2020.3 2021-02-04

For Support questions about any InterSystems products, contact:

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

Table of Contents

List of Figures

List of Tables

About This Book

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 document describes the scaling capabilities of InterSystems IRIS® data platform. Read this document if you are:

- Actively planning and implementing InterSystems IRIS configurations to meet specific needs within the enterprise.
- Seeking to understand the scaling features of InterSystems IRIS as they relate to your enterprise's existing and future needs.

Chapters in this guide include the following:

- InterSystems IRIS Scalability Overview
- Vertically Scaling InterSystems IRIS
- Horizontally Scaling for User Volume with Distributed Caching
- Horizontally Scaling for Data Volume with Sharding

1

InterSystems IRIS Scalability Overview

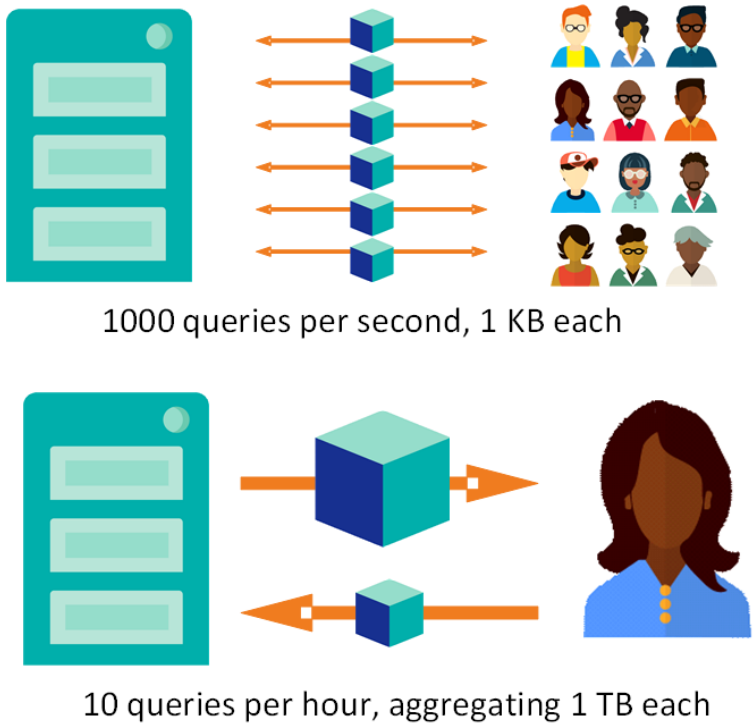
This chapter reviews the scaling features of InterSystems IRIS data platform, and provides guidelines for a first-order evaluation of scaling approaches for your enterprise data platform. Subsequent chapters cover each feature in more detail.

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.

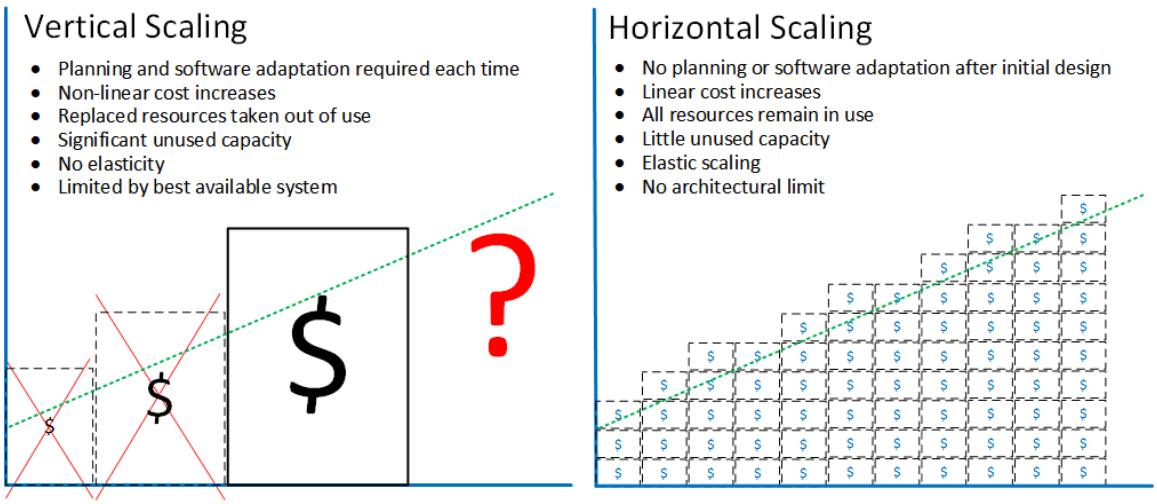
For more information on vertically scaling InterSystems IRIS data platform, see the chapter "Vertically Scaling InterSystems IRIS"

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

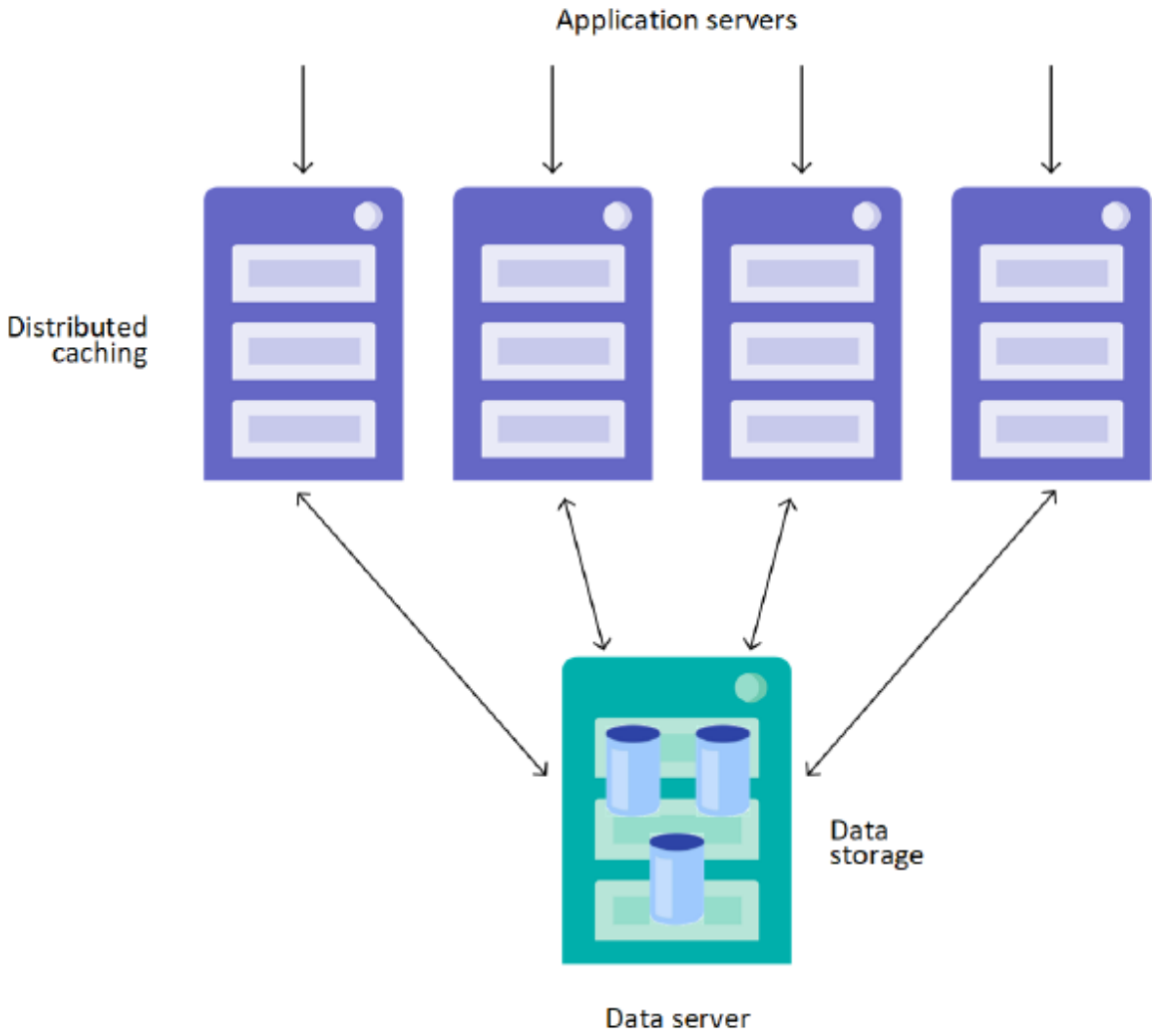
How can you scale horizontally when user volume is getting too big to handle with a single system at an acceptable cost? The short answer is to divide the user workload by connecting different users to different cluster nodes that handle their requests.

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.

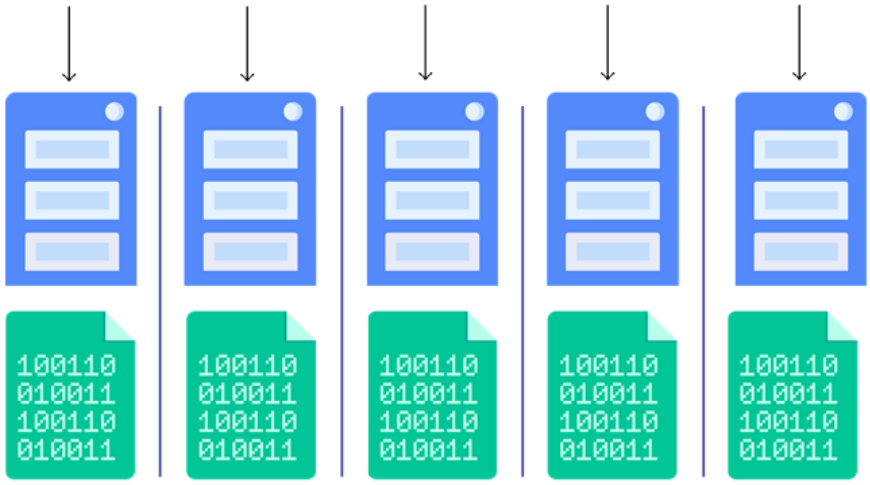
For more information on horizontally scaling InterSystems IRIS data platform for user volume, see the chapter "Horizontally Scaling for User Volume with Distributed Caching"

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 like Apache Spark.

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 the chapter "Horizontally Scaling for Data Volume with Sharding"

1.3.3 Using InterSystems Cloud Manager to Deploy Horizontally Scaled Configurations

InterSystems recommends using InterSystems Cloud Manager (ICM) to deploy InterSystems IRIS, including both distributed caching and sharded configurations. By combining plain text declarative configuration files, a simple command line interface, the widely-used Terraform infrastructure as code tool, and InterSystems IRIS deployment in containers, ICM provides you with a simple, intuitive way to provision cloud or virtual infrastructure and deploy the desired InterSystems IRIS architecture on that infrastructure, along with other services. ICM can significantly simplify the deployment process, especially for complex horizontal cluster configurations.

For more information on using ICM to deploy InterSystems IRIS, see the *InterSystems Cloud Manager Guide*.

1.4 Evaluating Your Workload for InterSystems IRIS Scaling Solutions

The subsequent chapters of this guide 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 table below summarizes the overview in this chapter, and provides some general guidelines concerning the scaling approach that might be of the most benefit in your current circumstances.

Table 1–1: InterSystems IRIS Scaling Solutions

Scaling Approach	Conditions	Possible Solutions	Pros (+) and Cons (-)
Vertical	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.	+ • Architectural simplicity • Hardware finely tuned to workload – • Nonlinear price/performance ratio • Persistent hardware limitations • Careful initial sizing required • One-way scaling only
	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.	
Horizontal	High multiuser query volume: frequent queries from large number of users.	Deploy application server configuration (distributed caching).	+ • More linear price/performance ratio • Can leverage commodity, virtual and cloud-based systems • Elastic scaling – • Emphasis on networking
	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)	

2

Vertically Scaling InterSystems IRIS

Scaling a system vertically by increasing its capacity and resources is a common, well-understood practice. Recognizing this, InterSystems IRIS includes a number of built-in capabilities that help you leverage the gains. Some operate transparently, while others require specific adjustments on your part to take full advantage.

This chapter discusses how to calculate the memory and CPU requirements of a server hosting an InterSystems IRIS instance and application, both initially and after collecting benchmarking and load testing results and information from existing sites, and how to take the best advantage of vertically scaling by increasing system memory or the CPU core count. In some cases, you may use these guidelines to evaluate whether a system that was chosen based on other criteria (such as corporate standards and cloud budget limits) is roughly sufficient to handle your workload requirements, whereas in others you may use them to plan the system you need based on those requirements. Additional actions that may improve performance are also discussed.

2.1 Memory Management and Scaling for InterSystems IRIS

Memory management is a critical element in optimizing performance and availability. For procedures for allocating memory within InterSystems IRIS, see Memory and Startup Settings in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.

2.1.1 Memory Overview

The goal of memory planning and management is to provide enough memory to all of the entities that use it under all normal operating circumstances. This is a critical factor in both performance and availability.

Generally, there are four main consumers of memory on a server hosting an InterSystems IRIS instance. At a high level, you can calculate the amount of physical memory required by simply adding up the requirements of each of the items on the following list:

- Operating system, including the file system cache
- Running applications, services, and processes other than InterSystems IRIS and the application based on it

  The memory needs of other entities processes running on the system can vary widely. If possible, make realistic estimates of the memory to be consumed by the software that will be cohosted with InterSystems IRIS.
- InterSystems IRIS and application processes

InterSystems IRIS is process-based. If you look at the operating system statistics while your application is running, you will see numerous processes running as part of InterSystems IRIS.

- InterSystems IRIS shared memory, which includes
  - The database cache (also known as the global buffer pool) and
  - The routine cache
  - The generic memory heap (gmheap)
  - Other shared memory structures

For the best possible performance, all four of these should be maintained in physical (system) memory under all normal operating conditions. Virtual memory and mechanisms for using it such as swap space and paging are important because they enable the system to continue operating during a transient memory capacity problem, but the highest priority is to include enough physical memory to avoid the use of virtual memory.

2.1.2 Calculating Memory Requirements and Allocation

Of course, every application is different and any given system may require a series of adjustments to optimize memory use. However, the following provides general guidelines to use as a basis in sizing memory for your InterSystems IRIS-based application. Benchmarking and performance load testing the application will further influence your estimate of the ideal memory sizing and parameters.

Important: If you have not configured sufficient physical memory on a Linux system and thus regularly come close to capacity, you run the risk that the out of memory killer may misidentify long-running InterSystems IRIS processes that touch a lot of memory in normal operation, such as the write daemon and CSP server processes, as the source of the problem and terminate them. *This will result in an outage of the InterSystems IRIS instance and require crash recovery at the subsequent startup*. Disabling the out of memory killer is *not recommended*, however, as this safety mechanism keeps your operating system from crashing when memory runs short, giving you a chance to intervene and restore InterSystems IRIS to normal operation. The recommended way to avoid this problem is to configure enough physical memory to avoid any chance of the out of memory killer coming into play. (For a detailed discussion of process memory in InterSystems IRIS, see Process Memory in InterSystems Products.)

Plan the initial memory requirements and allocation for your system using these guidelines:

- Calculate initial system memory to be installed in a physical system or provisioned in a virtual system as follows:
  - Estimate the memory required for the first two purposes cited in the previous section: operating system including the file system cache, and other installed programs.
  - For up to 1000 typical InterSystems IRIS processes and shared memory, add 4 to 8 GB per CPU core (physical or virtual).

    This core count does not include any threads such as Intel HyperThreading (HT) or IBM Simultaneous Multi-Threading (SMT) (see General Performance Enhancement on InterSystems IRIS Platforms). So, for example, if you have an IBM AIX LPAR with 8 cores allocated, the calculation would be 4-8 GB * 8 = 32 to 64 GB of total RAM allocated to that LPAR, even with SMT-4 enabled (which would appear as 32 logical processors).

    Bear in mind that the number of InterSystems IRIS processes running and their memory needs can vary significantly, and that shared memory requirements are influenced by various factors, including in particular the size of the database cache, which is typically larger for query-intensive workloads (including those hosted on distributed cache clusters and sharded clusters, as described in the following two chapters of this guide). In production, you can review the instance's use of shared memory and process memnory as follows:

    - To view the instance's shared memory usage, go to Management Portal's Shared Memory Heap Usage page (**System Operation > System Usage**, then click the **Shared Memory Heap Usage** button); for more information, see Generic (Shared) Memory Heap Usage in the *Monitoring Guide*.
    - To roughly estimate the maximum possible memory usage by Intersystems IRIS processes, multiply the the peak number of running processes by the default Maximum Per-Process Memory (bbsiz) setting of 262.144 MB, or the setting's actual value if it has been modified (see Setting the Maximum per Process Memory in the *System Administration Guide*). To learn more about memory use by InterSystems IRIS processes, see Process Memory in InterSystems Products.

- Allocate shared memory within InterSystems IRIS as follows:
  - On servers with less than 64 GB of RAM, allocate
    - 50% of total memory to the database cache
    - 256 MB minimum to the routine cache
    - 256 MB minimum to the generic memory heap
  - On servers with more than 64 GB of RAM, allocate
    - 70% of total memory to the database cache
    - 512 MB minimum to the routine cache
    - 384 MB minimum to the generic memory heap

Important: If System Monitor (described in Using System Monitor in the *Monitoring Guide*) generates the alert **Updates may become suspended due to low available buffers** or the warning **Available buffers are getting low (25% above the threshold for suspending updates)** while the system is under normal production workload, the database cache (global buffer pool) is not large enough and should be increased to optimize performance.

For swap space or the page file, as a general guideline, configure the smaller of a) 25 to 50% of your physical memory or b) 32 GB as virtual memory. As noted in Memory Overview, swapping and paging degrade performance and should come into play only when transient memory capacity problems require it. Further, you should configure alerts to notify operators when the system uses virtual memory so they can take immediate action to avoid more severe consequences.

Note: When large and huge pages are configured, as is highly recommended, InterSystems shared memory segments are pinned in physical memory and never swapped out; for more information, see Configuring Large and Huge Pages.

For procedures for allocating memory to the routine and database caches, configuring the generic memory heap, and setting the maximum memory per process, see Memory and Startup Settings in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.

Note: If you are configuring a data server in a distributed cache cluster, see Increase Data Server Database Caches for ECP Control Structures in the "Horizontally Scaling for User Volume with Distributed Caching" chapter of this guide for important information about adjustments to database cache sizes that may be necessary.

2.1.3 Vertically Scaling for Memory

Performance problems in production systems are often due to insufficient memory for application needs. Adding memory to the server hosting one or more InterSystems IRIS instances lets you allocate more to the database cache, the routine

cache, generic memory, or some combination. A database cache that is too small to hold the workload's working set forces queries to fall back to disk, greatly increasing the number of disk reads required and creating a major performance problem, so this is often a primary reason to add memory. Increases in generic memory and the routine cache may also be helpful under certain circumstances.

2.1.4 Configuring Large and Huge Pages

Where supported, the use of large and huge memory pages can be of significant performance benefit and is highly recommended, as described in the following:

- IBM AIX® — The use of large pages is highly recommended, especially when configuring over 16GB of shared memory (the sum of the database cache, the routine cache, and the generic memory heaps, as discussed in Calculating Memory Requirements and Allocation).

  By default, when large pages are configured, the system automatically uses them in memory allocation. If shared memory cannot be allocated in large pages, it is allocated in standard (small) pages. However, you can use the memlock parameter for finer-grained control over large pages.

  For more information, see Configuring Large Pages on IBM AIX® in the "Preparing to Install" chapter of the *Installation Guide* and memlock in the *Configuration Parameter File Reference*.

- Linux (all distributions) — The use of static huge pages (2MB) when available is highly recommended for either physical (bare metal) servers or virtualized servers. Using static huge pages for the InterSystems IRIS shared memory segments yields an average CPU utilization reduction of approximately 10-15% depending on the application.

  By default, when huge pages are configured, InterSystems IRIS attempts to provision shared memory in huge pages on startup. If there is not enough space, InterSystems IRIS reverts to standard pages and orphans the allocated huge page space, potentially causing system paging. However, you can use the memlock parameter to control this behavior and fail at startup if huge page allocation fails.

  For more information, see Configuring Huge Pages on Linux in the "Preparing to Install" chapter of the *Installation Guide* and memlock in the *Configuration Parameter File Reference*.

- Windows

  The use of large pages is recommended to reduce page table entry (PTE) overhead.

  By default, when large pages are configured, InterSystems IRIS attempts to provision shared memory in large pages on startup. If there is not enough space, InterSystems IRIS reverts to standard pages. However, you can use the memlock parameter to control this behavior and fail at startup if large page allocation fails.

  For more information, see Configuring Large Pages on Windows in the "Preparing to Install" chapter of the *Installation Guide* and memlock in the *Configuration Parameter File Reference*.

2.2 CPU Sizing and Scaling for InterSystems IRIS

InterSystems IRIS is designed to make the most of a system's total CPU capacity. Keep in mind that not all processors or processor cores are alike. There are variations at the surface such as clock speed, number of threads per core, and processor architectures, and also the varying impact of virtualization.

- Basic CPU Sizing
- Balancing Core Count and Speed
- Virtualization Considerations for CPU
- Leveraging Core Count with Parallel Query Execution

2.2.1 Basic CPU Sizing

Applications vary significantly from one to another, and there is no better measurement of CPU resource requirements than benchmarking and load testing your application and performance statistics collected from existing sites. If neither benchmarking or existing customer performance data is available, start with one of the following calculations:

- 1-2 processor cores per 100 users.
- 1 processor core for every 200,000 global references per second.

Important: These recommendations are only starting points when application-specific data is not available, and may not be appropriate for your application. It is very important to benchmark and load test your application to verify its exact CPU requirements.

2.2.2 Balancing Core Count and Speed

Given a choice between faster CPU cores and more CPU cores, consider the following:

- The more processes your application uses, the greater the benefit of raising the core count to increase concurrency and overall throughput.
- The fewer processes your application uses, the greater the benefit of the fastest possible cores.

For example, an application with a great many users concurrently running simple queries will benefit from a higher core count, while one with relatively fewer users executing compute-intensive queries would benefit from faster but fewer cores. In theory, both applications would benefit from many fast cores, assuming there is no resource contention when multiple processes are running in all those cores simultaneously. As noted in Calculating Memory Requirements and Allocation, the number of processor cores is a factor in estimating the memory to provision for a server, so increasing the core count may require additional memory.

2.2.3 Virtualization Considerations for CPU

Production systems are sized based on benchmarks and measurements at live customer sites. Virtualization using shared storage adds very little CPU overhead compared to bare metal, so it is valid to size virtual CPU requirements from bare metal monitoring.

Note: For hyper-converged infrastructure (HCI) deployments, add 10% to your estimated host-level CPU requirements to cover the overhead of HCI storage agents or appliances.

In determining the best core count for individual VMs, strike a balance between the number of hosts required for availability and minimizing costs and host management overhead; by increasing core counts, you may be able to satisfy the former requirement without violating the latter.

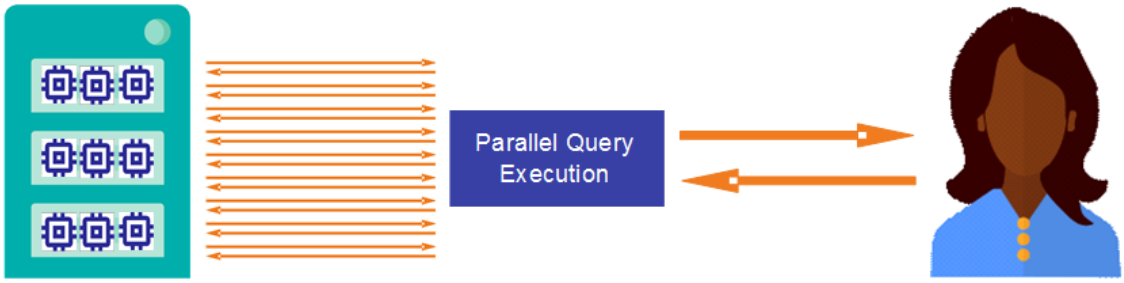
The following best practices should be applied to virtual CPU allocation:

- Production systems, especially database servers, are assumed to be highly utilized and should therefore be initially sized based on assumed equivalence between a physical CPU and its virtual counterpart. If you need six physical CPUs, assume you need six virtual CPUs.
- Do not allocate more vCPUs than required to optimize performance. Although large numbers of vCPUs can be allocated to a virtual machine, there can be a (usually small) performance overhead for managing unused vCPUs. The key here is to monitor your systems regularly to ensure that vCPUs are correctly allocated.

2.2.4 Leveraging Core Count with Parallel Query Execution

When you upgrade by adding CPU cores, an InterSystems IRIS feature called parallel query execution helps you take the most effective advantage of the increased capacity.

Figure 2–1: Parallel Query Execution



Parallel query execution is built on a flexible infrastructure for maximizing CPU usage that spawns one process per CPU core, and is most effective with large data volumes, such as analytical workloads that make large aggregations.

For more information on parallel query processing, see Parallel Query Processing in the “Optimizing Query Performance “ chapter of the *SQL Optimization Guide*.

2.3 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 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.

3

Horizontally Scaling for User Volume with Distributed Caching

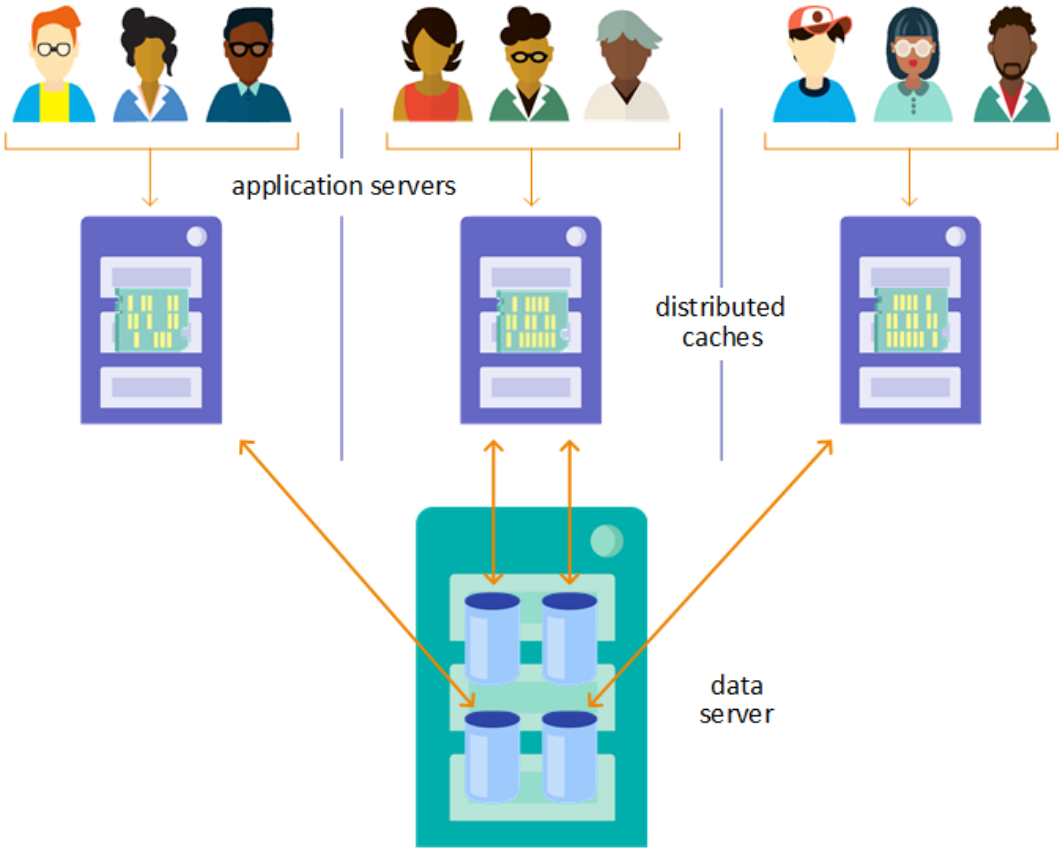
When vertical scaling alone proves insufficient for scaling your InterSystems IRIS data platform to meet your workload's requirements, you can consider *distributed caching*, an architecturally straightforward, application-transparent, low-cost approach to horizontal scaling.

3.1 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 3–1: 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 distibuted caching:

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

3.1.1 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 3–2: Local databases mapped to local namespaces on a single InterSystems IRIS instance

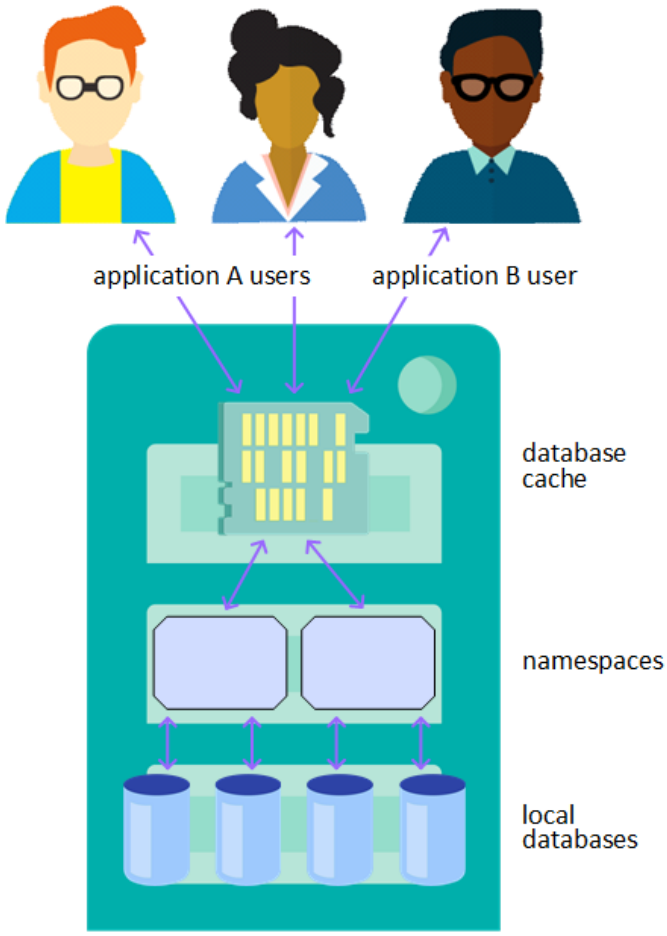
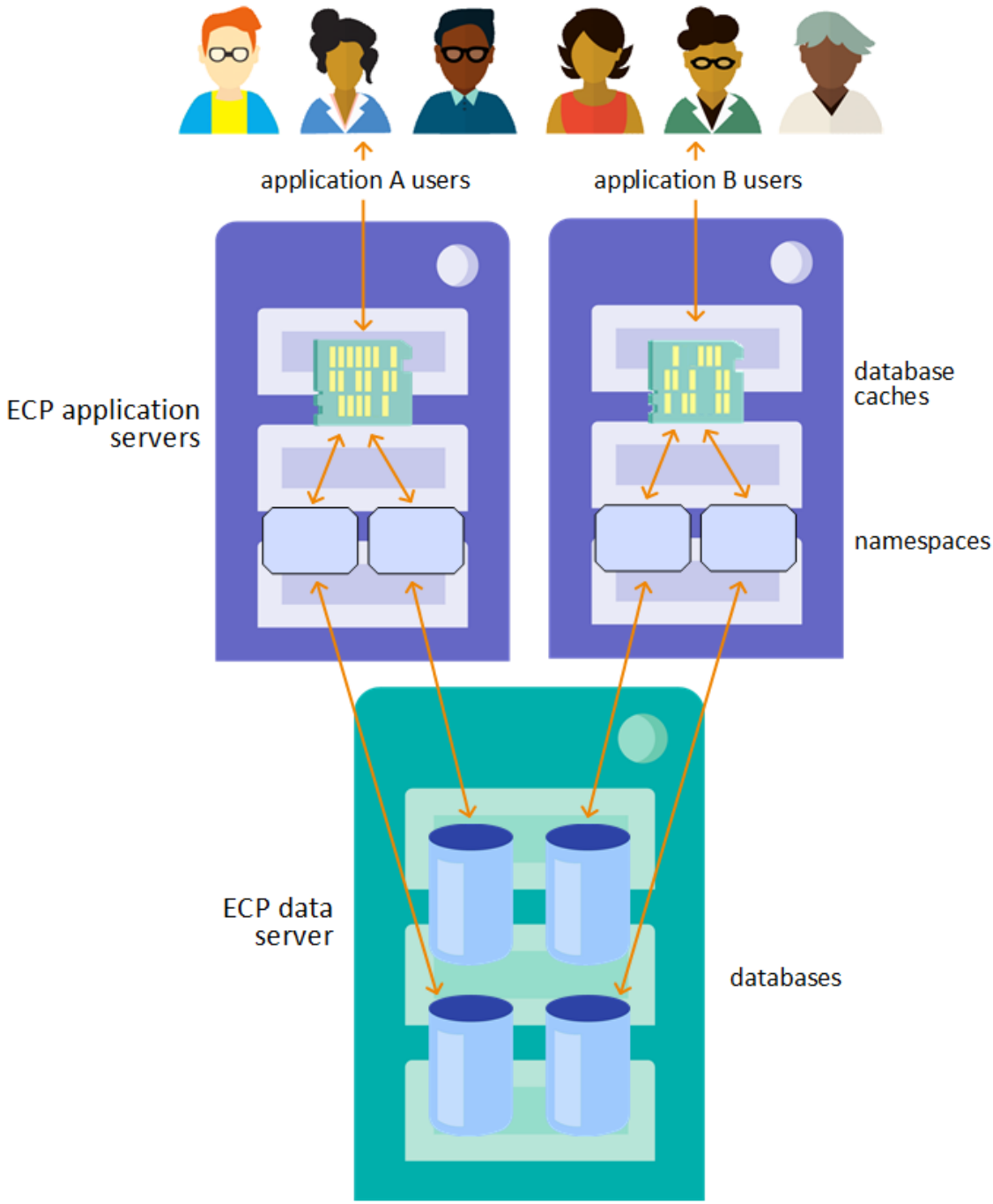


Figure 3–3: Remote databases on a data server mapped to namespaces on application servers in a distributed cache cluster



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.

3.1.2 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.

3.1.3 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.

3.1.4 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 mir-

roring in a distributed cache cluster, see Configuring ECP Connections to a Mirror the "Configuring Mirroring" chapter in the *High Availability Guide*.

The other failover strategies detailed in the "System Failover Strategies" chapter of the *High Availability Guide* 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.

3.2 Deploying a Distributed Cache Cluster

An InterSystems IRIS distributed cache cluster consists of a data server providing data to one or more application servers, which in turn provide it to the application. This section describes procedures for deploying a distributed cache cluster.

Note: For an important discussion of performance planning, including memory management and scaling, CPU sizing and scaling, and other considerations, see the "Vertical Scaling" chapter of this guide.

HealthShare Health Connect does not support distributed caching.

The recommended method for deploying InterSystems IRIS data platform is InterSystems Cloud Manager (ICM). By combining plain text declarative configuration files, a simple command line interface, the widely-used Terraform infrastructure as code tool, and InterSystems IRIS deployment in containers, ICM provides you with a simple, intuitive way to provision cloud or virtual infrastructure and deploy the desired InterSystems IRIS architecture on that infrastructure, along with other services. ICM can deploy distributed cache clusters and other InterSystems IRIS configurations on Amazon Web Services, Google Cloud Platform, Microsoft Azure or VMware vSphere. ICM can also deploy InterSystems IRIS on an existing physical or virtual cluster.

Deploy the Cluster with InterSystems Cloud Manager offers an overview of the process of using ICM to deploy the distributed cache cluster.

Note: For a brief introduction to ICM including a hands-on exploration of deploying InterSystems IRIS, see First Look: InterSystems Cloud Manager. For complete ICM documentation, see the *InterSystems Cloud Manager Guide*.

You can also deploy a distributed cluster by using the Management Portal to configure existing or newly installed InterSystems IRIS instances; instructions for this procedure are provided in Deploy the Cluster Using the Management Portal.

Information about securing the cluster after deployment is provided in Distributed Cache Cluster Security.

Note: The most typical distributed cache cluster configuration involves one InterSystems IRIS instance per system, and one cluster role per instance — that is, either data server or application server. When deploying using ICM, this configuration is the only option. The provided procedure for using the Management Portal assumes this configuration as well.

3.2.1 Data Server/Application Server Compatibility

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 (see Using the NLS Settings Page of the Management Portal in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*).

3.2.2 Deploy the Cluster with InterSystems Cloud Manager

There are several stages involved in provisioning and deploying a containerized InterSystems IRIS configuration, including a distributed cache cluster, with ICM. The *ICM Guide* provides complete documentation of ICM, including details of each of the stages. This section briefly reviews the stages and provides links to the *ICM Guide*.

- Launch ICM
- Obtain Security-Related Files
- Define the deployment
- Provision the infrastructure
- Deploy and manage services
- Unprovision the infrastructure

3.2.2.1 Launch ICM

ICM is provided as a container image. With the exception of InterSystems IRIS licenses and security-related files as described, everything required by ICM to carry out its provisioning, deployment, and management tasks is included in the ICM container, including a /Samples directory that provides you with samples of the elements required by ICM, customized to the supported cloud providers. To launch ICM, on a system on which Docker is installed, you use the **docker run** command with the ICM image from the InterSystems Container Registry to start the ICM container.

For detailed information about launching ICM, see Launch ICM in the "Using ICM" chapter of the *ICM Guide*.

3.2.2.2 Obtain Security-Related Files

Before defining your deployment, you must obtain security-related files including cloud provider credentials and keys for SSH and TLS. For more information about these files and how to obtain them, see Obtain Security-Related Files in the "Using ICM" chapter.

3.2.2.3 Define the Deployment

ICM uses JSON files as input. To provide the needed parameters to ICM, you must represent your target configuration and the platform on which it is to be deployed in two of ICM's JSON configuration files: the defaults.json file, which contains information about the entire deployment, and the definitions.json file, which contains information about the types and numbers of the nodes provisioned and deployed by ICM, as well as details specific to each node type. For example, the defaults file determines which cloud provider your distributed cache cluster nodes are provisioned on and the locations of the required security files and InterSystems IRIS license keys, while the definitions file determines how many application servers are included in the sharded cluster and whether the data volume for the data server will be larger than for the application servers. Most ICM parameters have defaults; a limited number of parameters can be specified on the ICM command line as well as in the configuration file.

For sample defaults and definitions files for distributed cache cluster deployment, see Define the Deployment in the "Using ICM" chapter of the *ICM Guide*. You can create your files by adapting the template defaults.json and definitions.json files provided with ICM in the /Samples directory (for example, /Samples/AWS for AWS deployments), or start with the contents of the samples provided in the documentation. For a complete list of the fields you can include in these files, see ICM Configuration Parameters in the "ICM Reference" chapter of the *ICM Guide*.

ICM includes the node types DM and AM for provisioning and deploying a cluster's data and application servers, and such types as WS (web server) and LB (load balancer) for associated systems. When the DM node is mirrored, you can deploy an AR node as mirror arbiter. For detailed descriptions of the node types (for use in the Role field in the definitions file) that ICM can provision, configure, and deploy services on, see ICM Node Types in the "ICM Reference" chapter of the *ICM Guide*.

Note: When deploying InterSystems IRIS containers with ICM, including those in a distributed cache cluster, you can override one or more InterSystems IRIS configuration settings for all of the containers, or override different settings for the InterSystems IRIS containers on different node types, such as the DM and AM nodes; for more information, see Deploying with Customized InterSystems IRIS Configurations in the "ICM Reference" chapter of the *ICM Guide*.

3.2.2.4 Provision the Infrastructure

When your definitions files are complete, begin the provisioning phase by issuing the command **icm provision** on the ICM command line. This command allocates and configures the nodes specified in the definitions file. At completion, ICM also provides a summary of the nodes and associated components that have been provisioned, and outputs a command line which can be used to delete the infrastructure at a later date, for example:

```
Machine           IP Address      DNS Name
-------           ----------       -------
ACME-DM-TEST-0001  00.53.183.209   ec2-00-53-183-209.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0002  00.56.59.42     ec2-00-56-59-42.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003  00.67.1.11      ec2-00-67-1-11.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0004  00.193.117.217  ec2-00-193-117-217.us-west-1.compute.amazonaws.com
ACME-LB-TEST-0000  (virtual AM)    ACME-AM-TEST-1546467861.amazonaws.com
To destroy: icm unprovision [-cleanUp] [-force]
```

Once your infrastructure is provisioned, you can use several infrastructure management commands. For detailed information about these and the **icm provision** command, including reprovisioning an existing configuration to scale out or in or to modify the nodes, see Provision the Infrastructure in the "Using ICM" chapter of the *ICM Guide*.

3.2.2.5 Deploy and Manage Services

ICM carries out deployment of InterSystems IRIS and other software services using Docker images, which it runs as containers by making calls to Docker. In addition to Docker, ICM also carries out some InterSystems IRIS-specific configuration over JDBC. There are many container management tools available that can be used to extend ICM's deployment and management capabilities.

The **icm run** command downloads, creates, and starts the specified container on the provisioned nodes. The **icm run** command has a number of useful options, and also lets you specify Docker options to be included, so there are many versions on the command line depending on your needs. Here are just two examples:

- When deploying InterSystems IRIS images, you must set the password for the predefined accounts on the deployed instances. The simplest way to do this is to omit a password specification from both the definitions files and the command line, which causes ICM to prompt you for the password (with typing masked) when you execute **icm run**. But this may not be possible in some situations, such as when running ICM commands with a script, in which case you need either the **-iscPassword** command line option or the iscPassword field in the defaults file.
- You can deploy different containers on different nodes — for example, InterSystems IRIS on the DM and AM nodes and the InterSystems Web Gateway on the WS nodes — by specifying different values for the DockerImage field (such as intersystems/iris:stable and intersystems/webgateway:stable) in the different node definitions in the definitions.json file. To deploy multiple containers on a node or nodes, however, you can run the **icm run** command more than once — the first time to deploy the image(s) specified by the DockerImage field, and subsequent times using the **-image** and **-container** options (and possibly the **-role** or **-machine** option) to deploy a custom container.

For a full discussion of the use of the **icm run** command, including redeploying services on an existing configuration, see The icm run Command in the "Using ICM" chapter of the *ICM Guide*.

At deployment completion, ICM sends a link to the appropriate node's Management Portal, for example:

```
Management Portal available at: http://ec2-00-153-49-109.us-west-1.compute.amazonaws.com:52773/csp/sys/UtilHome.csp
```

In the case of a distributed cache cluster, the provided link is for the data server instance.

Once your containers are deployed, you can use a number of ICM commands to manage the deployed containers and interact with the containers and the InterSystems IRIS instances and other services running inside them; for more information, see Container Management Commands and Service Management Commands in the "Using ICM" chapter of the *ICM Guide*.

3.2.2.6 Unprovision the Infrastructure

Because public cloud platform instances continually generate charges and unused instances in private clouds consume resources to no purpose, it is important to unprovision infrastructure in a timely manner. The **icm unprovision** command deallocates the provisioned infrastructure based on the state files created during provisioning. As described in Provision the Infrastructure, the needed command line is provided in the **icm provision** output when the provisioning phase is complete, and is also contained in the ICM log file, for example:

```
To destroy: icm unprovision [-cleanUp] [-force]
```

This command line includes any configuration file location override options (**-definitions**, **-defaults**, **-instances**, or **-stateDir**) that you included in the **icm provision** command, as these are required to successfully unprovision.

For more detailed information about the unprovisioning phase, see Unprovision the Infrastructure in the "Using ICM" chapter of the *ICM Guide*.

3.2.3 Deploy the Cluster Using the Management Portal

Once you have installed 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**).

3.2.3.1 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 the "Services" chapter of the *Security Administration Guide*.

- 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 ECP service can also be enabled and the maximum number of application servers set using the EnableECP and MaxServerConn settings in the configuration parameter file (CPF), including in a CPF 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.

3.2.3.2 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.
- Add 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 and enable the ECP service. 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 the IPv6 Support section in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.

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 checkbox if the data server is the primary failover member in a mirror. (See Configuring Application Server Connections to a Mirror in the "Configuring Mirroring" chapter of the *High Availability Guide* for important information about configuring a mirror primary as a data server.)
- **Use SSL/TLS** — Use this checkbox 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 checkbox; 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 checkbox 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 checkbox.

  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 namepsace.
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** checkbox.

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 in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.) 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.

3.2.4 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 the Available Services section of the "Services" chapter of the *Security Administration Guide*.)

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 the "Managed Key Encryption" chapter of the *Security Administration Guide*.

3.2.4.1 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 the "Using TLS with InterSystems IRIS" chapter of the *Security Administration 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** checkbox, 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-wide Security Parameters page (**System Administration > Security > System Security > System-wide Security Parameters**) and select **Enabled** for the **Superserver SSL/TLS support** setting.
- 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 the section Creating or Editing a TLS Configuration in the "Using TLS with InterSystems IRIS" chapter of the *Security Administration Guide*. 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 "Using TLS with InterSystems IRIS" chapter of the *Security Administration 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 in the "Using TLS with InterSystems IRIS" chapter of the *Security Administration Guide*, 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.)

3.2.4.2 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.

3.2.4.3 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 in the "About InterSystems Security" chapter of the *Security Administration Guide*.

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 the "Using LDAP" chapter of the *Security Administration 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.

3.2.4.4 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.

3.3 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

3.3.1 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.

3.3.1.1 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 the ECP Connection States section.

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 the ECP Connection Operations section for more information.

Delete

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

3.3.1.2 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 the ECP Connection States section.

Client IP

The host name or IP address of the application server

IP Port

The port number used to connect to the application server.

3.3.2 ECP Connection States

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

Table 3–1: 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

3.3.2.1 Application Server Connection States

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

- Not Connected
- Connection in Progress
- Normal
- Trouble
- Transitional Recovery
- Disabled

Application Server Not Connected State

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

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 Normal State

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

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

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.

Application Server Disabled State

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.

3.3.2.2 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*.

3.3.3 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.

3.4 Developing Distributed Cache Applications

This chapter 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 chapter discusses the following topics:

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

3.4.1 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 in the *Configuration Parameter File Reference*.

Table 3–2: 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.

3.4.2 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:

```
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.

3.4.3 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
- The $Increment Function and Application Counters

3.4.3.1 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.

3.4.3.2 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 in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.

3.4.3.3 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.

3.4.3.4 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 in "ECP Recovery Guarantees and Limitations".

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.

3.4.3.5 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.

3.4.3.6 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.

3.4.3.7 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**.

3.4.4 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.

3.4.4.1 <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.

3.4.4.2 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.

3.5 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 the ECP Recovery Limitations section of the aforementioned appendix.

3.5.1 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.

3.5.1.1 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.

3.5.1.2 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.

3.5.1.3 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.

3.5.1.4 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.

3.5.1.5 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.

3.5.1.6 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.

3.5.1.7 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*.

3.5.1.8 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 the ECP and Clusters $Increment Limitation section 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.

3.5.1.9 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.

3.5.1.10 $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 the ECP and Clusters $Increment Limitation section for more information.

3.5.1.11 ECP Sync Method Guarantee

Process A updates a global located on Data Server S, and issues a **$system.ECP.Sync()** to S. Process B then issues a **$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.

3.5.2 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

3.5.2.1 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.

3.5.2.2 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.

3.5.2.3 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.

3.5.2.4 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.

3.5.2.5 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.

3.5.2.6 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.

3.5.2.7 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.

3.5.2.8 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.

3.5.2.9 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.

3.5.2.10 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.

3.5.2.11 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 he 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

4

Horizontally Scaling for Data Volume with Sharding

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

4.1 Overview of InterSystems IRIS Sharding

Sharding is a significant horizontal scalability feature of InterSystems IRIS data platform. 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 First Look: Scaling for Data Volume with an InterSystems IRIS Sharded Cluster.

4.1.1 Elements of Sharding

Horizontally scaling InterSystems IRIS through 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.

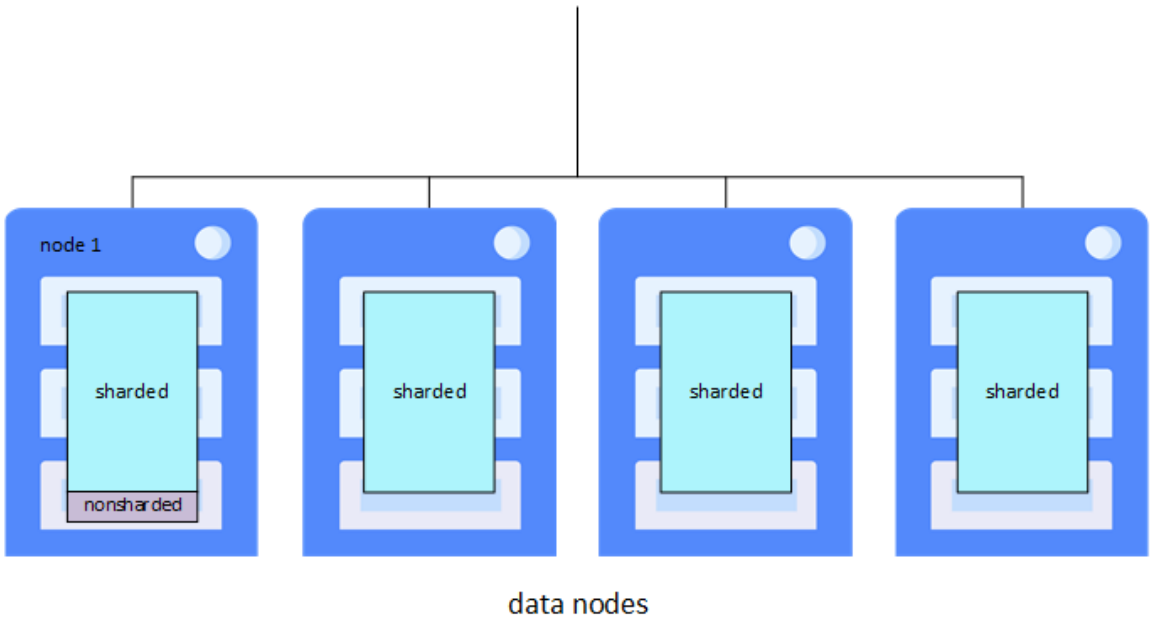
- Because the data partitions are hosted by separate instances, each has a dedicated cache to serve its own partition of the data set, rather than a single instance's cache serving the entire data set.

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 like Apache Spark.

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 4–1: 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.

4.1.2 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.

4.1.3 Namespace-level Sharding Architecture

Previous 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 master application 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.

The %SYSTEM.Sharding API and the Sharding Configuration page of the Management Portal remain available for use in sharded cluster deployment based on the namespace-level architecture; see Deploying the Namespace-level Architecture for procedures.

4.2 Deploying the Sharded Cluster

This section provides procedures for deploying a basic InterSystems IRIS sharded cluster consisting of nonmirrored data nodes.

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 by reprovisioning your ICM deployment, using the %SYSTEM.Cluster API, or configuring or deploying them using CPF settings. 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, 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.

Note: For an important discussion of performance planning, including memory management and scaling, CPU sizing and scaling, and other considerations, see the "Vertical Scaling" chapter of this guide.

HealthShare Health Connect does not support sharding.

There are several ways to deploy a sharded cluster, as follows:

- With InterSystems Cloud Manager (ICM), you can automatically deploy any InterSystems IRIS data platform configuration. By combining plain text declarative configuration files, a simple command line interface, the widely-used Terraform infrastructure as code tool, and InterSystems IRIS deployment in containers, ICM provides you with a simple, intuitive way to provision cloud or virtual infrastructure and deploy the desired InterSystems IRIS architecture on that infrastructure, along with other services.

  Deploy the Cluster with InterSystems Cloud Manager offers an overview of the process of using ICM to deploy the sharded cluster. (For a brief introduction to ICM that includes a hands-on exploration of deploying a sharded cluster, see First Look: InterSystems Cloud Manager. For complete ICM documentation, see the *InterSystems Cloud Manager Guide*.)

- You can manually deploy a sharded cluster using the %SYSTEM.Cluster API. For instructions, see Deploy the Cluster Using the %SYSTEM.Cluster API.

- You can deploy a sharded cluster using configuration parameter file (CPF) settings; for details, see Configure or Deploy the Cluster Using CPF Settings.

- You can use the InterSystems Kubernetes Operator (IKO) to deploy sharded clusters on any Kubernetes platform; for details, see Using the InterSystems Kubernetes Operator.

Note: In the most typical sharded cluster configuration, each cluster node consists of one InterSystems IRIS instance on one physical or virtual system. When deploying using ICM, this configuration is the only option, and it is assume in the provided procedures for deploying with the %SYSTEM.Cluster API and CPF settings.

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 the "Using LDAP" chapter of the *Security Administration Guide*.

Regardless of the method you use to deploy the 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, as listed in the following:

- Plan data nodes
- Estimate the database cache and database sizes

- Deploy the cluster using InterSystems Cloud Manager
- Deploy the cluster using the %SYSTEM.Cluster API
- Configure or deploy the cluster using CPF settings

4.2.1 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. Since you can always add data nodes to an existing cluster and rebalance the sharded data (see Add Data Nodes and Rebalance Data), 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 the "Vertical Scaling" chapter of this guide.)

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 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 tools like ICM can help ensure this consistency.

The general recommended best practice is to load balance application connections across all of the data nodes in a cluster. ICM can automatically provision and configure a load balancer for the data nodes as needed when deploying in a public cloud; if deploying a sharded cluster by other means, a load balancing mechanism is required.

4.2.2 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 ICM, you can specify these settings by including properties in the configuration files. When you deploy manually using the Sharding API, you can specify database cache size before configuring the sharded cluster, and specify database settings in your calls. Both 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.

4.2.2.1 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.

4.2.2.2 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.

4.2.3 Deploy the Cluster Using InterSystems Cloud Manager

There are several stages involved in provisioning and deploying a containerized InterSystems IRIS configuration, including a sharded cluster, with ICM. The *ICM Guide* provides complete documentation of ICM, including details of each of the stages. This section briefly reviews the stages and provides links to the *ICM Guide*.

- Launch ICM
- Obtain Security-Related Files
- Define the deployment
- Provision the infrastructure
- Deploy and manage services
- Unprovision the infrastructure

4.2.3.1 Launch ICM

ICM is provided as a container image. With the exception of InterSystems IRIS licenses and security-related files as described, everything required by ICM to carry out its provisioning, deployment, and management tasks is included in the ICM container, including a /Samples directory that provides you with samples of the elements required by ICM, customized to the supported cloud providers. To launch ICM, on a system on which Docker is installed, you use the **docker run** command with the ICM image from the InterSystems Container Registry to start the ICM container.

For detailed information about launching ICM, see Launch ICM in the "Using ICM" chapter of the *ICM Guide*.

4.2.3.2 Obtain Security-Related Files

Before defining your deployment, you must obtain security-related files including cloud provider credentials and keys for SSH and TLS. For more information about these files and how to obtain them, see Obtain Security-Related Files in the "Using ICM" chapter.

4.2.3.3 Define the Deployment

ICM uses JSON files as input. To provide the needed parameters to ICM, you must represent your target configuration and the platform on which it is to be deployed in two of ICM's JSON configuration files: the defaults.json file, which contains

information about the entire deployment, and the definitions.json file, which contains information about the types and numbers of the nodes provisioned and deployed by ICM, as well as details specific to each node type. For example, the defaults file determines which cloud provider your sharded cluster nodes are provisioned on and the locations of the required security files and InterSystems IRIS license keys, while the definitions file determines how many data nodes and compute nodes are included in the sharded cluster and the specifications of their hosts. Most ICM parameters have defaults; a limited number of parameters can be specified on the ICM command line as well as in the configuration file.

For sample defaults and definitions files for sharded cluster deployment, see Define the Deployment in the "Using ICM" chapter of the *ICM Guide*. You can create your files by adapting the template defaults.json and definitions.json files provided with ICM in the /Samples directory (for example, /Samples/AWS for AWS deployments), or start with the contents of the samples provided in the documentation. For a complete list of the fields you can include in these files, see ICM Configuration Parameters in the "ICM Reference" chapter of the *ICM Guide*.

For a complete list of the fields you can include in these files, see ICM Configuration Parameters in the "ICM Reference" chapter of the *ICM Guide*.

Note: All InterSystems IRIS instances in a sharded cluster must have sharding licenses.

ICM includes the node types DATA and COMPUTE for provisioning and deploying a cluster's data and compute nodes along with the AR node type for arbiters for mirrored clusters, as well as such types as WS (web server) and LB (load balancer) for associated systems. Node types representing nodes in the namespace-level architecture are also included. For detailed descriptions of the node types (for use in the Role field in the definitions file) that ICM can provision, configure, and deploy services on, see ICM Node Types in the "ICM Reference" chapter of the *ICM Guide*.

When creating your configuration files, bear in mind that they must represent not only the number of data nodes you want to include but their database cache and database sizes, which you determined in Estimate the Database Cache and Database Sizes. This can be accomplished as follows:

- Database cache size — Every InterSystems IRIS instance, including those running in the containers deployed by ICM, is installed with a predetermined set of configuration settings, recorded in its configuration parameters file (CPF). The UserCPF field specifies a CPF merge file containing one or more of these configuration settings; you can customize all of the InterSystems IRIS instances you deploy by including it in your defaults file. You can also customize settings for a specific node type by including it in the node definition. For example, to allocate a database cache of 150 GB of 8-kilobyte blocks on each data node, you would customize the value of the globals CPF setting by including UserCPF in the DATA node definition to specify a CPF merge file containing the following:

  ```
  [config]
  globals=0,0,150000,0,0,0
  ```

  A sample CPF merge file, /Samples/cpf/iris.cpf, is included in the ICM container, and all of the sample defaults.json files contain the UserCPF property, specifying this file. For information about using a merge file to override CPF settings, see Deploying with Customized InterSystems IRIS Configurations in the "ICM Reference" chapter of the *ICM Guide*.

  Of course, the cloud nodes you provision as data nodes must have sufficient memory to accommodate the target database cache size. The instance type you specify in the defaults file, or in the DATA node definition in the definitions file, determines the characteristics of the provisioned cloud nodes that become data nodes, including memory. The name of this field is different for each cloud provider; the equivalent fields for the four cloud providers are shown in the table that follows (see Provider-Specific Parameters in the *ICM Guide* for more information).

Provider	Field	For information see
AWS	InstanceType	Amazon EC2 Instance Types in the AWS documentation
GCP	MachineType	Machine types in the GCP documentation
Azure	Size	Sizes for virtual machines in Azure in the Azure documentation
Tencent	InstanceType	Instance Types in the Tencent documentation
vSphere	Memory	Provider-Specific Parameters in the *ICM Guide*

Important: The larger a cloud instance type or storage volume is, the more it costs. It is therefore advisable to size as accurately as possible, without wasting capacity.

- Database size — The DataVolumeSize property (see General Parameters in the "ICM Reference" chapter of the *ICM Guide*) determines the size of the deployed InterSystems IRIS instance's storage volume for data, which is where the default globals databases for the master and shard namespaces are located. This setting must be large enough to accommodate the target size of the default globals database, as described in Estimate the Database Cache and Database Sizes. In the case of AWS and GCP, this setting is limited by the field DataVolumeType (see Provider-Specific Parameters in the *ICM Guide*).

  Note: In some cases, it may be advisable to increase the size of the generic memory heap on the cluster members, which can be done by using a CPF merge file to override the gmheap setting. For information about allocating memory to the generic memory heap, see in the *Configuration Parameter File Reference*.

It is important is that you determine their values for the database cache and globals database sizes based on your particular situation, and include them as needed. In general, appropriate sizing of the data nodes in a sharded cluster and configuration of the InterSystems IRIS instances on them is a complex matter influenced by many factors, including experience; as your experience accumulates, you are likely to modify some of these settings.

4.2.3.4 Provision the Infrastructure

When your definitions files are complete, begin the provisioning phase by issuing the command **icm provision** on the ICM command line. This command allocates and configures the nodes specified in the definitions file. At completion, ICM also provides a summary of the nodes and associated components that have been provisioned, and outputs a command line which can be used to delete the infrastructure at a later date, for example:

```
Machine              IP Address      DNS Name
-------              ----------      --------
ACME-DATA-TEST-0001  00.53.183.209   ec2-00-53-183-209.us-west-1.compute.amazonaws.com
ACME-DATA-TEST-0002  00.56.59.42     ec2-00-56-59-42.us-west-1.compute.amazonaws.com
ACME-DATA-TEST-0003  00.67.1.11      ec2-00-67-1-11.us-west-1.compute.amazonaws.com
ACME-DATA-TEST-0004  00.193.117.217  ec2-00-193-117-217.us-west-1.compute.amazonaws.com
ACME-LB-TEST-0000    (virtual DATA)  ACME-LB-TEST-1546467861.amazonaws.com
To destroy: icm unprovision [-cleanUp] [-force]
```

Once your infrastructure is provisioned, you can use several infrastructure management commands. For detailed information about these and the **icm provision** command, including reprovisioning an existing configuration to scale out or in or to modify the nodes, see Provision the Infrastructure in the "Using ICM" chapter of the *ICM Guide*.

4.2.3.5 Deploy and Manage Services

ICM carries out deployment of InterSystems IRIS and other software services using Docker images, which it runs as containers by making calls to Docker. In addition to Docker, ICM also carries out some InterSystems IRIS-specific configuration over JDBC. There are many container management tools available that can be used to extend ICM's deployment and management capabilities.

The **icm run** command downloads, creates, and starts the specified container on the provisioned nodes. The **icm run** command has a number of useful options, and also lets you specify Docker options to be included, so there are many versions on the command line depending on your needs. Here are just two examples:

- When deploying InterSystems IRIS images, you must set the password for the predefined accounts on the deployed instances. The simplest way to do this is to omit a password specification from both the definitions files and the command line, which causes ICM to prompt you for the password (with typing masked) when you execute **icm run**. But this may not be possible in some situations, such as when running ICM commands with a script, in which case you need either the **-iscPassword** command line option or the iscPassword field in the defaults file.
- You can deploy different containers on different nodes — for example, InterSystems IRIS on the DM and AM nodes and the InterSystems Web Gateway on the WS nodes — by specifying different values for the DockerImage field (such as intersystems/iris:stable and intersystems/webgateway:stable) in the different node definitions in the definitions.json file. To deploy multiple containers on a node or nodes, however, you can run the **icm run** command more than once — the first time to deploy the image(s) specified by the DockerImage field, and subsequent times using the **-image** and **-container** options (and possibly the **-role** or **-machine** option) to deploy a custom container.

For a full discussion of the use of the **icm run** command, including redeploying services on an existing configuration, see The icm run Command in the "Using ICM" chapter of the *ICM Guide*.

At deployment completion, ICM sends a link to the appropriate node's Management Portal, for example:

```
Management Portal available at: http://ec2-00-153-49-109.us-west-1.compute.amazonaws.com:52773/csp/sys/UtilHome.csp
```

In the case of a sharded cluster, the provided link is for node 1.

Once your containers are deployed, you can use a number of ICM commands to manage the deployed containers and interact with the containers and the InterSystems IRIS instances and other services running inside them; for more information, see Container Management Commands and Service Management Commands in the "Using ICM" chapter of the *ICM Guide*.

4.2.3.6 Unprovision the Infrastructure

Because public cloud platform instances continually generate charges and unused instances in private clouds consume resources to no purpose, it is important to unprovision infrastructure in a timely manner. The **icm unprovision** command deallocates the provisioned infrastructure based on the state files created during provisioning. As described in Provision the Infrastructure, the needed command line is provided when the provisioning phase is complete, and is also contained in the ICM log file, for example:

```
To destroy: icm unprovision [-cleanUp] [-force]
```

For more detailed information about the unprovisioning phase, see Unprovision the Infrastructure in the "Using ICM" chapter of the *ICM Guide*.

4.2.4 Deploy the Cluster Using the %SYSTEM.Cluster API

Use the following procedure to deploy a basic InterSystems IRIS sharded cluster of data nodes using the %SYSTEM.Cluster API. You will probably find it useful to refer the %SYSTEM.Cluster class documentation in the *InterSystems Class Reference*.

Note: As with all classes in the %SYSTEM package, the %SYSTEM.Cluster methods are available through $SYSTEM.Cluster.

- Provision or identify the infrastructure
- Deploy InterSystems IRIS on the data nodes
- Configure the data nodes

This procedure assumes you are deploying the InterSystems IRIS instances on the nodes before you begin configuring the cluster, but can be adapted for use with existing instances.

Note: This procedure does not cover the deployment of mirrored data nodes; that procedure is provided in Mirror for High Availability. Similarly, see Deploy Compute Nodes for information about using the API to add compute nodes to a basic cluster.

4.2.4.1 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.

4.2.4.2 Deploy InterSystems IRIS on the Data Nodes

This procedure assumes that each system hosts or will host a single InterSystems IRIS instance.

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 "File System Recommendations" and "Storage Recommendations" in the "Preparing to Install" chapter of the *Installation Guide* and Journaling Best Practices in the "Journaling" chapter of the *Data Integrity Guide*.

The configuration of all 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 in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*; 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 CPF merge file.

In some cases, it may also be advisable to increase the size of the generic memory heap on the cluster members. The generic memory heap can be configured either using the Management Portal, as described for the gmheap parameter in the *Configuration Parameter File Reference*, 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 generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

4. Ensure that the instance has both the MaxServerConn and MaxServers parameters set to values at least as great as the planned number of nodes in the sharded cluster. These values can be viewed and changed using the **Maximum number of application servers** and **Maximum number of data servers** settings on the ECP Settings page of the Management Portal (**System Administration > Configuration > Connectivity > ECP Settings**), or by editing the instance's configuration parameter file (CPF). On UNIX® and Linux platforms, the instances can be deployed or installed with the desired values using a CPF merge file.

Important: If you change any of the parameters described in the above procedure by editing an instance's CPF, you must restart the instance after doing so.

4.2.4.3 Configure the Data Nodes

For each instance in the cluster and its host, perform the steps described for its role within the cluster.

- Configure node 1
- Configure the remaining data nodes

Note: Remember that the calls described in this procedure do not work with mirrored nodes; for information on deploying mirrored data nodes, see Mirror for High Availability.

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.

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 in *Running InterSystems Products in Containers* and 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 for the instance, see InterSystems IRIS Connection Information in *InterSystems IRIS Basics: Connecting an IDE*.)

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

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.

4.2.5 Configure or Deploy the Cluster Using CPF Settings

Every InterSystems IRIS instance is installed and operates with a file in the installation directory named iris.cpf, which contains most of its configuration settings. The instance reads this *configuration parameter file*, or CPF, at startup to obtain the values for these settings. One of the tasks you can accomplish by modifying the settings in an instance's CPF is to configure it as a member of a sharded cluster.

There are two ways to use CPF settings to create a sharded cluster, as follows

- Configure existing instances by manually modifying their CPFs

  Assuming they are installed on an appropriately networked group of hosts, you can configure a group of existing instances as a sharded cluster by modifying their iris.cpf files to set the needed parameters, and then restarting them so that the new settings can take effect. As described in Configure Node 1 in "Deploy the Cluster Using the %SYSTEM.Cluster API", node 1 must be separately configured before other data nodes, so you must modify the node 1 instance's CPF and restart it before the remaining data nodes.

- Deploy new instances with customized CPFs

  On UNIX® and Linux platforms, you can deploy a cluster by customizing the CPF of each instance before it starts and reads its settings for the first time by using the CPF merge feature. You do this by using the **ISC_CPF_MERGE_FILE** environment variable to specify a separate file containing one or more settings to be merged into the CPF with which a new instance is installed or deployed; this allows you to deploy multiple instances with differing CPFs from the same source without having to manually modify each CPF individually, supporting automated deployment and a DevOps approach. Use of this feature is described in the following documentation:

  – Create and Use a CPF Merge File in the "Introduction to the Configuration Parameter File" chapter of the *Configuration Parameter File Reference*.

  – Deploying Customized InterSystems IRIS Instances in *Running InterSystems Products in Containers.*

Because the instance with the CPF merge file configuring it 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 as the remaining data nodes.

The procedure provided here can be used either to configure existing instances as a cluster or deploy new instances as a cluster by first configuring or deploying data node 1 and then configuring or deploying the remaining data nodes. If the names of the hosts of the cluster nodes match or will match a predictable pattern, you can also configure or deploy the cluster using a hostname pattern. The CPF settings named in the following steps are linked to their respective entries in the *Configuration Parameter File Reference*.

Important: Whether you are configuring existing InterSystems IRIS instances or preparing the infrastructure on which to deploy new instances, be sure to review provision or identify the infrastructure and deploy InterSystems IRIS on the data nodes in Deploy Cluster Using the %SYSTEM.Cluster API for requirements and best practices for the InterSystems IRIS instances in a sharded cluster and their hosts.

Note: This procedure does not cover the deployment of mirrored data nodes; instructions for this are provided in Mirror Data Nodes for High Availability. Similarly, see Deploy Compute Nodes for information about using CPF settings to add compute nodes to a basic cluster.

4.2.5.1 Configure or Deploy Data Node 1

Data node 1 must be configured or deployed first. The following table includes the CPF settings required, to be modified for an existing instance or included in the CPF merge file for an instance you are deploying. Once the instance has been modified and restarted or deployed, you can verify that the settings are as desired by viewing the iris.cpf file.

Table 4–1: CPF settings for data node 1

Section	Setting	Description	Value for data node 1
[Startup]	ShardRole	Determines the node's role in the cluster.	node1
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the target cache size you determined for data nodes, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	You probably need to increase this setting from the default of 37.5 MB to optimize the performance of the data nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

Configure an existing instance as data node 1

To configure an existing instance as node 1, edit and modify the *install-dir*/iris.cpf file to insert the `[Startup]` setting and update the `[config]` settings as shown below, then restart the instance.

```
[ConfigFile]
Product=IRIS
Version=2020.1

[Databases]
...
[Startup] (insert the following settings]
ShardRole=node1
...
[config] (modify the following settings)
...
MaxServerConn=64
MaxServers=64
...
globals=0,0,204800,0,0,0
gmheap=393,216
```

Note: The globals setting of `0,0,204800,0,0,0` illustrated in the example illustrates an 8–kilobyte buffer database cache of 200 GB.

The gmheap setting of 393,216 KB illustrated in the example represents the minimum recommended generic memory heap size, 384 MB, for systems with over 64 GB of memory.

Deploy an instance as data node 1

To deploy node 1 using the CPF merge feature, prepare and specify a CPF merge file with the following contents; the merge file contains only the settings that will be changed in the default CPF before the instance's initial startup. (The values shown for the `[config]` settings are examples.)

```
[Startup]
ShardRole=node1
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

4.2.5.2 Configure or deploy or the remaining data nodes

Once node 1 has been a) configured and restarted or b) deployed and successfully started, the other data node instances can be configured and restarted or deployed in any order. For each of the remaining data nodes, use the settings indicated in Deploy or configure data node 1, with two modifications:

- Set ShardRole to **data** rather than node1.
- Add ShardClusterURL to the [Startup] section, set to the cluster URL of node 1.

The following table and CPF excerpt explain and illustrate these settings:

Table 4–2: CPF settings for remaining data nodes

Section	Setting	Description	Value for remaining data nodes
[Startup]	ShardRole (**CHANGE**)	Determines the node's role in the cluster.	**data**
	ShardClusterURL (**ADD**)	Identifies the existing node (typically node 1) to use as a template when adding a data node to the cluster, as described in Configure the remaining data nodes in the %SYSTEM.Cluster API procedure.	Hostname and superserver port of node 1 in the form IRIS://*host*:*port*.
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the target cache size you determined for data nodes, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	The default is 37.5 MB; you probably need to increase this setting from the default to optimize the performance of the data nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

```
[Startup]
ShardRole=data
ShardClusterURL=IRIS://datanode1:1972
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

Configure the remaining existing data node instances by updating the indicated settings in their iris.cpf files and restarting them, or deploy the remaining data node instances using a CPF merge file (as illustrated above) to customize the settings. Once the instance has been deployed or modified and restarted, you can verify that the settings are as desired by viewing the iris.cpf file.

4.2.5.3 Configure or Deploy the Cluster Using a Hostname Pattern

If the names of the existing hosts you want to configure as a sharded cluster match a predictable pattern, or the new hosts you are provisioning for the cluster will do so, you can determine the cluster roles of the instances you configure or deploy based on those names. This approach allows you to use the same CPF modifications or CPF merge file for all instances,

although they must still be configured in the right order. To do this, on all nodes, use the settings indicated in Configure or Deploy Data Node 1, with these modifications:

- Set ShardRole to **auto** rather than to node1.
- Add ShardMasterRegexp to the [Startup] section (see table and example).
- Add ShardRegexp to the [Startup] section (see table and example).
- Remove ShardClusterURL (not included when using a hostname pattern).

The following table and CPF excerpt explain and illustrate these settings:

Table 4–3: CPF Settings when using a hostname pattern

Section	Setting	Description	Value for remaining data nodes

Section	Setting	Description	Value for remaining data nodes
[setup]	ShardRole (**CHANGE**)	Determines the node's role in the cluster; when set to AUTO, the node to configure as node 1 is identified by matching its hostname to the regular expression specified by ShardMasterRegexp, while the others are configured as additional data nodes. Cannot be set to auto unless ShardMasterRegexp is specified.	**auto**
	ShardMasterRegexp (**ADD**)	When ShardRole=auto, identifies the instance to configure as node 1 by matching the name of the instance's host to the regular expression specified as its value. For example, if you specify ShardRole=auto and ShardMasterRegexp=-0$ for instances on hosts named **`data-0`**, **`data-1`**, and **`data-2`**, **`data-0`** is configured as node 1 and the others as additional data nodes. If no hostnames match the specified regular expression, or more than one does, none of the nodes are configured as data nodes.	*regular_expression* matching cluster node hostnames
	ShardRegexp (**ADD**)	When ShardRole=AUTO, validates that the hostname of the instance matches the regular expression provided. For example, if ShardRole=AUTO, ShardMasterRegexp=-0$, and ShardRegexp=-[0-9]+$ are specified for instances on hosts named **`data-0`**, **`data-11`**, **`data-22`**, and **`data-2b`**, the instances are configured as follows: • **`data-0`** becomes node 1 (it matches ShardMasterRegexp) • **`data-11`** and **`data-22`** become additional data nodes (they don't match ShardMasterRegexp but do match ShardRegexp) • **`data-2b`** does not join the cluster (it matches neither regular expression)	*regular_expression* matching cluster node hostnames

Section	Setting	Description	Value for remaining data nodes
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the target cache size you determined for data nodes, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	You probably need to increase this setting from the default of 37.5 MB to optimize the performance of the data nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

```
[Startup]
ShardRole=data
ShardMasterRegexp=-0$
ShardRegexp=-[0-9]$
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

Configure and restart node 1 or deploy it with a CPF merge file as illustrated, confirm that it is running, then configure the remaining existing data node instances by updating the indicated settings in their iris.cpf files and restarting, or deploy the remaining data node instances using the same CPF merge file. At any stage you can optionally verify an instance's settings by viewing its iris.cpf file.

4.3 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

4.3.1 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.

4.3.2 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.

4.3.2.1 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.

4.3.2.2 Evaluate Unique Constraints

When a sharded table has a unique constraint (see Field Constraint and Unique Fields Constraint in the "Create Table" entry in the *InterSystems SQL Reference*), 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.

4.3.2.3 Create the Tables

Create the empty sharded tables using standard CREATE TABLE statements (see CREATE TABLE in the *SQL Reference*) 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 in the "Create Table" entry in the *SQL Reference*, 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 stream fields, 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 in the *InterSystems SQL Reference*.

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 in the "Defining Tables" chapter of *Using InterSystems SQL*. 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, such as the use of stream properties, will be introduced in upcoming versions of InterSystems IRIS.

4.3.3 Load Data Onto the Cluster

Data can be loaded into sharded tables by using INSERT statements through any InterSystems IRIS interface that supports SQL, for example the Management Portal, the Terminal, or JDBC. Rapid bulk loading of data into sharded tables is supported by the transparent parallel load capability built into the InterSystems IRIS JDBC driver, as well as by the InterSystems IRIS Connector for Spark, which leverages the same capability. Java-based applications also transparently benefit from the InterSystems IRIS JDBC driver's parallel loading capability.

4.3.3.1 Load Data Using INSERT

You can verify that a sharded table was created as intended by loading data using an INSERT or INSERT SELECT FROM through any InterSystems IRIS interface that supports SQL and then querying the table or tables in question.

4.3.3.2 Load Data Using the InterSystems IRIS Spark Connector

The InterSystems IRIS Spark Connector allows you to add Apache Spark capabilities to a sharded cluster. The recommended configuration is to locate Spark worker nodes on the data node hosts and a Spark master node on the node 1 host, connected to the corresponding InterSystems IRIS instances. For more information about the Spark Connector and using it to load data, see *Using the InterSystems IRIS Spark Connector*.

4.3.3.3 Load Data Using the InterSystems IRIS JDBC Driver

Using the transparent parallel load capability of the InterSystems IRIS JDBC driver, you can construct a tool that retrieves data from a data source and passes it to the target table on the sharded cluster by means of JDBC connections, as follows:

- Any database for which a suitable JDBC driver is available can act as the data source.
- The InterSystems IRIS JDBC driver, which has been optimized for the parallel insertion of large numbers of records into the shards of a sharded table, is used to connect to the target table on the sharded cluster. (See *Using Java JDBC with InterSystems IRIS* for complete information about the InterSystems IRIS JDBC driver.)

  Note: For most data loading operations, including simple INSERTs, 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. For example, when connecting from a local client to a sharded cluster provisioned in the cloud by ICM, the data node IP addresses known to and returned by the shard master will be on the cloud subnet and thus inaccessible from the local machine.

For your convenience, InterSystems provides the Simple Data Transfer utility, a Java command-line utility for massive data transfer from a JDBC data source or CSV file to a JDBC-compliant database. While the utility works with any supported target or source, it is optimized to work with InterSystems IRIS as the target, and is intended primarily for extremely fast relocation of huge datasets. The utility works with both nonsharded and sharded namespaces, and takes full advantage of parallelization when the target table is sharded. For more information about Simple Data Transfer, see "Using the Simple Data Transfer Utility".

4.3.4 Create and Load Nonsharded Tables

You can create nonsharded tables in the master namespace on the shard master data server, 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.

4.4 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, or the corresponding namespace on a shard master app server, 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.
- Queries with stream fields in the SELECT list are not currently supported.
- 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 a nonaggregated field and an aggregate function, unless the GROUP BY clause is used.
  - 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. For more information about purging cached queries, see Purging Cached Queries in the "Cached Queries" chapter of the *SQL Optimization Guide*.

4.5 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).

4.5.1 Add Data Nodes and Rebalance Data

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, you may want to increase the number of data nodes because the size of your sharded data may grow significantly enough to make a higher shard count desirable, for example, or because a resource constraint has been removed. Data nodes can be added by reprovisioning and redeploying the cluster using ICM (see Reprovisioning the Infrastructure and Redeploying Services in the *ICM Guide*), or by repeating the relevant steps outlined in Deploy the cluster using the %SYSTEM.Cluster API or Configure or deploy the cluster using CPF settings.

When you have added data nodes to a cluster but have not yet rebalanced the data, as described later in this section, distribution of sharded data on the cluster is as follows:

- The rows already contained in sharded tables that existed before the nodes were added are not redistributed, but remain where they were on the original set of nodes.
- When sharded data is added to the cluster after new data nodes are added, in the form of either rows added to a previously existing table or a new table created and loaded with data, their distribution depends on each table's shard key, as follows:
  - If the existing or new table has a system assigned shard key (SASK), the new rows are evenly distributed across all of the data nodes, including the new nodes.
  - If the existing or new table has a user defined shard key (UDSK), the new rows are distributed across the original set of data nodes only, and not to the newly-added nodes.

    Note: If there were no existing UDSK tables before the new nodes were added, rows in UDSK tables created after the nodes are added are distributed across all nodes.

To take full advantage of the new nodes, you must evenly distribute all sharded data on the cluster across all of the nodes. To accomplish this, you can use the **$SYSTEM.Sharding.Rebalance()** API call, which rebalances existing sharded data across the expanded set of data nodes.

For example, if you go from four data nodes to eight, rebalancing takes you from four existing data nodes with one fourth of the sharded data on each, plus four empty new servers, to eight servers with one eighth of the data on each. After rebalancing, both rows added to existing tables and the rows of newly created tables are also distributed across all eight data nodes, regardless of shard key. Thus, once you have rebalanced, all sharded data — including existing tables, rows added to existing tables, and new tables — is evenly distributed across all eight data nodes.

Rebalancing cannot coincide with queries and updates, and so can take place only when the sharded cluster is offline and no other sharded operations are possible. (In a future release, this limitation will be removed.) For this reason, the **$SYSTEM.Sharding.Rebalance()** call places the sharded cluster in a state in which queries and updates of sharded tables are not permitted to execute, and return an error if attempted.

Each rebalancing call can specify a time limit, however, so that the call can be scheduled in a maintenance window, move as much data as possible within the window, and return the sharded cluster to a fully-usable state before the window ends. By using this approach with repeated calls, you can fully rebalance the cluster over a series of scheduled maintenance outages

without otherwise interfering with its operation. 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.

Note: Query and update operations execute correctly before rebalancing is performed (when new data nodes are still empty), in between the calls of a multicall rebalancing operation, and after rebalancing is complete, but they are most efficient after all of the data has been rebalanced across all of the data nodes.

The illustration that follows show the process of adding shards and rebalancing data using a multicall rebalancing operation.

Figure 4–2: Adding a Shard and Rebalancing Data

4.5.2 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 and make 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.

You can deploy a mirrored sharded cluster using ICM, the %SYSTEM.Cluster API, or CPF settings, as described in the following. However you deploy the cluster, the recommended best practice is to load balance application connections across all of the mirrored data nodes in the cluster.

You can also convert an existing nonmirrored cluster to a mirrored cluster.

If you make mirroring changes outside of the %SYSTEM.Cluster API, for example using the Management Portal or the SYS.Mirror API, you must call **$SYSTEM.Sharding.VerifyShards()** to update the mirrored cluster's metadata after doing so.

Note: You also can use the InterSystems Kubernetes Operator IKO) to deploy a sharded cluster with compute nodes on any Kubernetes platform; for details, see Using the InterSystems Kubernetes Operator.)

4.5.2.1 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 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**.

4.5.2.2 Deploy a Mirrored Cluster Using ICM

To deploy a fully mirrored using InterSystems Cloud Manager, refer to Define the Deployment and make the following changes:

1. Add **"Mirror": "true"** to the defaults file.
2. If you want to configure a mirror arbiter, include an AR node in the definitions file.
3. Specify a number of DATA nodes that matches the value of MirrorMap in the DATA node definition, as follows:
   - If the MirrorMap field is omitted, the default value of primary,backup is in effect, which means DATA nodes can provisioned only in even numbers, to be configured as failover pairs.
   - Including MirrorMap lets you include DR asyncs in the DATA node mirrors. For example, if MirrorMap is primary,backup,async, DATA nodes can be provisioned in multiples of three to be configured as mirrors that each include a failover pair and a DR async; if the value is primary,backup,async,async, DATA nodes can be provisioned in multiples of four to configure mirrors with two DR asyncs each, and so on. (The number of DATA nodes can also be fewer than the number of elements in the MirrorMap value, as long as it is at least two.)

   If you define a number of DATA nodes when Mirror is set to true that does not match the MirrorMap setting, provisioning fails. For complete information about the MirrorMap setting and its effect on ICM deployment of a mirrored sharded cluster, see Rules for Mirroring in the *ICM Guide*.

   Note: If you are deploying a namespace-level mirrored cluster, the MirrorMap field governs the deployment of DS node in the same way as it does DATA nodes in a node-level mirrored cluster.

   You can use MirrorMap in conjunction with the Zone and ZoneMap fields to distribute the members of each mirror across multiple cloud availability zones; for more information, see Deploying Across Multiple Zones in the *ICM Guide*.

For detailed information on deploying mirrored configurations with ICM, see ICM Cluster Topology and Mirroring in the "ICM Reference" chapter of the *ICM Guide*.

4.5.2.3 Deploy a Mirrored Cluster Using the %SYSTEM.Cluster API

The %SYSTEM.Cluster API can simultaneously apply data node configuration and mirror configuration to nonmirrored instances, but it also recognizes the existing mirror configurations of instances you select for configuration as data nodes. This means that you can either configure existing mirrors as data nodes or simultaneously configure nonmirrored instances as both data nodes and mirror members, as follows:

- If you start with an existing mirror primary, the API adds it to the cluster as a data node without any changes to its mirror configuration. You can then add the existing backup to the cluster by identifying it as the backup of that primary. If the first failover member you added does not have a failover partner, you can add a nonmirrored instance as the second failover member and the API automatically configures it as such before attaching it to the cluster.
- If you start with a nonmirrored instance, the API configures it as the first failover member of a mirror, based on the settings you provide in arguments, before adding it to the cluster as a data node. You can then add another nonmirrored instance by specifying it as the second failover member; the API automatically configures it as such before attaching it to the cluster.
- For any failover pair that you have configured as a data node, you can optionally add existing DR async mirror members to the data node, or configure nonmirrored instances as DR asyncs in the data node mirror.

Given this flexibility, you may find it convenient to configure all of the mirrors before deploying them as a sharded cluster. The Management Portal and the %SYSTEM.MIRROR API allow you to specify more settings than the %SYSTEM.Cluster calls described in the following procedure; see the "Configuring Mirroring" chapter of the *High Availability Guide* for

details. Even if you plan to use %SYSTEM.Cluster API calls to configure the mirrors, it is a good idea to review the procedures and settings in Creating a Mirror in the "Configuring Mirroring" chapter before you do so.

The recommended best practice is that the mirrored sharded cluster be fully configured, 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 with data on it to a mirrored cluster.)

The procedure for deploying a mirrored cluster is similar to that for deploying a nonmirrored cluster. First, provision or identify the infrastructure and install InterSystems IRIS on the cluster nodes as described in Deploy the Cluster Using the %SYSTEM.Cluster API. Then, to configure the node 1 mirror (first data node) and then configure the remaining mirrored data nodes, use the following procedure.

Note: You cannot use this procedure 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.

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()**, as described in Configure Node 1) in "Deploy the Cluster Using the %SYSTEM.Cluster API"; 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 an 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, see InterSystems IRIS Connection Information in *InterSystems IRIS Basics: Connecting an IDE*.)

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 in "Deploy the Cluster Using the %SYSTEM.Cluster API", 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
```

4.5.2.4 Configure or Deploy a Mirrored Cluster Using CPF Settings

For a general understanding of the use of CPF settings to deploy a sharded cluster or to configure existing InterSystems IRIS instances as a sharded cluster, review the procedures in Configure or Deploy the Cluster Using CPF Settings.

To configure or deploy a mirrored cluster using CPF settings, you must identify either the existing InterSystems IRIS instances you will configure as a sharded cluster or the infrastructure on which you will deploy the cluster. Whichever is the case, be sure to review both Provision or Identify the Infrastructure and Deploy InterSystems IRIS on the Data Nodes in Deploy the Cluster Using the %SYSTEM.Cluster API for important requirements and recommendations concerning both the infrastructure that hosts the data node mirrors and the InterSystems IRIS instances that are the mirror members.

There are two ways to configure or deploy a mirrored cluster using CPF settings, as follows:

- Configure or deploy the data node mirrors manually

  The manual method of configuring or deploying mirrored data nodes requires several different sets of CPF modifications or merge files, because different combinations of the ShardMirrorMember setting, which specifies the failover role (primary or backup) of the node, and ShardClusterURL must be applied to different nodes in the correct order.

- Configure or deploy the mirrored data nodes automatically using a host name pattern

  If the names of the existing hosts you want to configure as a sharded cluster match a predictable pattern, or the new hosts you are provisioning for the cluster will do so, you can determine which node becomes the node 1 primary, which becomes its backup, and which of the others become primaries and backups based on those names. This approach allows you to use the same CPF modifications or CPF merge file for all instances, although they must still be configured in the right order.

Configure or deploy the data node mirrors manually

To configure or deploy a mirrored cluster manually using CPF settings, follow these steps:

1. Configure and restart or deploy the node 1 primary using the procedure and settings indicated in Configure or Deploy Data Node 1 in "Configure or Deploy the Cluster Using CPF Settings", with this modification:

   - Add ShardMirrorMember to the [Startup] section.

   The following table and CPF excerpt explain and illustrate these settings:

   Table 4–4: CPF settings for data node 1 mirror primary

Section	Setting	Description	Value for data node 1
[Startup]	ShardRole	Determines the node's role in the cluster.	node1
	ShardMirrorMember (**ADD**)	Determines the node's role in the mirror.	primary
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the target cache size you determined for data nodes, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	You probably need to increase this setting from the default of 37.5 MB to optimize the performance of data nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

```
[Startup]
ShardRole=node1
ShardMirrorMember=primary
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

   Setting for the remaining nodes are provided separately in the steps that follow, but for a summary of the parameter values used to configure a mirrored cluster, see the table CPF parameters for configuring a mirrored sharded cluster after the last step.

2. When the node 1 primary has been configured and restarted, or has been deployed and has started successfully, configure or deploy the node 1 backup using the settings in the previous step, with these modifications:

   - Set ShardRole to **data** rather than node1.

- Add ShardClusterURL to the [Startup] section, specifying the node 1 primary as the template node (see illustration below for example), as described in Configure or Deploy the Remaining Data Nodes in "Configure or Deploy the Cluster Using CPF Settings".
- Set ShardMirrorMember to **backup** rather than primary.

The following CPF excerpt illustrates these settings:

```
[Startup]
ShardRole=data
ShardClusterURL=IRIS://primarynode1:1972
ShardMirrorMember=backup
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

When the node 1 backup has been configured and restarted, or has been deployed and has started successfully, you have created the node 1 mirror.

To configure or deploy a DR async member of the node 1 mirror, use the settings for the backup, above, but set ShardMirrorMember to **async** rather than backup. You can configure as many as 14 DR async members of a mirror.

3. Configure or deploy the primaries of the remaining data nodes using the settings indicated in the previous step, with this modification:

   - Set ShardMirrorMember to **primary** rather than backup or async.

   The following CPF excerpt illustrates these settings:

   ```
   [Startup]
   ShardRole=data
   ShardClusterURL=IRIS://primarynode1:1972
   ShardMirrorMember=primary
   [config]
   MaxServerConn=64
   MaxServers=64
   globals=0,0,204800,0,0,0
   gmheap=393,216
   ```

   After you have configured and restarted or deployed and successfully started the data node mirror primaries, proceed to the next step.

4. To configure or deploy a data node as a mirror backup, you must specify the mirror primary it is joining using the ShardClusterURL setting. For this reason, each of the remaining data nodes must be configured or deployed with a separate set of CPF modifications or CPF merge file. Do this using the settings indicated in the previous step, with these modifications:

   - Set ShardClusterURL to the URL of the primary to which the backup will be assigned rather than the node 1 primary (see illustration below for example), using the format described in Configure or Deploy the Remaining Data Nodes.
   - Set ShardMirrorMember to **backup** rather than primary.

   The following CPF excerpt illustrates these settings:

   ```
   [Startup]
   ShardRole=data
   ShardClusterURL=IRIS://primarynode4:1972
   ShardMirrorMember=backup
   [config]
   MaxServerConn=64
   MaxServers=64
   globals=0,0,204800,0,0,0
   gmheap=393,216
   ```

   When each data node backup has been configured and restarted, or has been deployed and has started successfully, you have created all of the remaining data node mirrors and the sharded cluster is complete.

To configure or deploy a DR async member of a data node mirror, use the settings for backups, above, but set ShardMirrorMember to **async** rather than backup. You can configure as many as 14 DR async members in each mirror.

The following table summarizes the use of the relevant parameters to configure the entire cluster of mirrored data nodes:

Table 4–5: CPF parameters for configuring a mirrored sharded cluster

hostname	ShardRole	ShardMirrorMember	ShardClusterURL	Node Configuration
data-0	node1	primary	n/a	Node 1 primary
data-1	data	backup	IRIS://data-0:1972	Backup of node 1 primary
data-1a (optional)		async	IRIS://data-0:1972	DR async of node 1
data-2		primary	IRIS://data-0:1972	Second data node primary
data-3		backup	IRIS://data-2:1972	Backup of primary on data-2
data-3a (optional)		async	IRIS://data-2:1972	DR async of second data node
data-4		primary	IRIS://data-0:1972	Third data node primary
data-5		backup	IRIS://data-4:1972	Backup of primary on data-4
data-5a (optional)		async	IRIS://data-4:1972	DR async of third data node

The data node primaries other than node 1 can be configured or deployed in any order; a backup or DR async cannot be configured or deployed until its primary is running.

Configure or deploy the data node mirrors automatically using a host name pattern

To configure or deploy a mirrored cluster automatically with CPF settings using a host name pattern, use the procedure and settings indicated in Configure or Deploy the Cluster Using a Hostname Pattern in "Configure or Deploy the Cluster Using CPF Settings", with these modifications:

- Add ShardMirrorMember to the `[Setup]` section
- Add ArbiterURL to the `[Startup]` section,

Note: You cannot configure or deploy DR async mirror members when using a hostname pattern. However, after deploying the failover pairs using a hostname pattern, you can add DR asyncs to the data node mirrors using the %SYSTEM.Cluster API, or by using a mirroring interface such as the Management Portal or the SYS.Mirror API and updating the cluster metadata for the changes after you have made them.

The following table and CPF excerpt explain and illustrate these settings:

Table 4–6: CPF Settings when using a hostname pattern

Section	Setting	Description	Value for remaining data nodes

Section	Setting	Description	Value for remaining data nodes
[Startup]	ShardRole	Determines the node's role in the cluster; when set to AUTO, the node to configure as node 1 is identified by matching its hostname to the regular expression specified by ShardMasterRegexp, while the others are configured as additional data nodes. Cannot be set to AUTO unless ShardMasterRegexp is specified.	AUTO
	ShardMirrorMember (**ADD**)	When used with ShardRole=AUTO, ShardMirrorMember=auto configures or deploys mirror failover roles based on the hostnames of the nodes on which the instances are deployed: if the integer following the final hyphen (-) in the hostname is even, the instance is configured as a primary, and if odd, as a backup. For example, if instances are deployed on four nodes with the hostnames **`data-0, data-1`**, **`data-2`**, and **`data-3`**, and ShardRole=AUTO, ShardMirrorMember=auto, and ShardMasterRegexp=-0$ are specified, the instances are configured as follows: • **`data-0`** becomes the node 1 primary (matches ShardMasterRegexp; digit after final hyphen in hostname is even) • **`data-1`** becomes the node 1 backup (hostname ends with odd digit and is next after node 1 primary hostname) • **`data-2`** becomes a data node primary (hostname ends with even digit) • **`data-3`** becomes the backup for the data node primary on **`data-2`** (hostname ends with odd digit)	auto
	ArbiterURL (**ADD**)	Identifies the arbiter used by the mirrors in a mirrored sharded cluster; include in any CPF merge file containing ShardMirrorMember to configure the specified mirrors with an arbiter.	*host*:*port*
	ShardMasterRegexp	When ShardRole=AUTO, identifies the instance to configure as node 1 by matching the name of the instance's host to the regular expression specified as its value; see ShardMirrorMember row for example. If no hostnames match the specified regular expression, or more than one does, none of the nodes are configured as data nodes.	*regular_expression* matching cluster node hostnames
	ShardRegexp	When ShardRole=AUTO, validates that the	*regular_expression* matching cluster

Section	Setting	Description	Value for remaining data nodes
		hostname of the instance matches the regular expression provided. For example, if ShardRegexp=-[0-9]$ and a fifth host called **data-2b** is added to the example provided for ShardMirrorMember above, **data-2b** is not configured as a data node because it matches neither ShardMasterRegExp (required for node 1) or ShardRegExp.(required for remaining data nodes).	node hostnames
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the target cache size you determined for data nodes, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	You probably need to increase this setting from the default of 37.5 MB to optimize the performance of data nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

```
[Startup]
ShardRole=AUTO
ShardMirrorMember=auto
ArbiterURL=arbiter:2188
ShardMasterRegexp=-0$
ShardRegexp=-[0-9]$
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

Configure and restart the node 1 primary or deploy it with a CPF merge file as illustrated, confirm that it is running, then configure the remaining existing data node instances by updating the same settings in their iris.cpf files and restarting or deploy the remaining data node instances using the same CPF merge file. At any stage you can optionally verify an instance's settings by viewing its iris.cpf file.

4.5.2.5 Convert a Nonmirrored Cluster to a Mirrored Cluster

You can convert an existing nonmirrored sharded cluster to a mirrored cluster using the procedure outlined in this section. 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 the section Deploy the Cluster Using the %SYSTEM.Cluster API before continuing. Familiarity with mirror configuration procedures, as described in the Configuring Mirroring chapter of the *High Availability Guide*, is also helpful but not required; the steps in this procedure provide links to that chapter where appropriate.

Note: With minor adaptations, you can use this procedure to convert a nonmirrored namespace-level cluster to a mirrored cluster. If doing so, review Deploying the Namespace-level Architecture before continuing.

1. Prepare the nodes that are to be added to the cluster as backup failover members according to the instructions in the first two steps of "Deploy the Cluster Using the %SYSTEM.Cluster API", 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 in the *High Availability Guide*).

   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.

2. Begin a maintenance window for the sharded cluster.
3. On each current data node, start the ISCAgent, then create a mirror and configure the first failover member.
4. To convert the cluster to a mirrored configuration, open the InterSystems Terminal for the instance on node 1 and in the master namespace (**IRISDM** by default) 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 in "Deploy the Cluster Using the %SYSTEM.Cluster API") 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).

5. 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 (**IRISCLUSTER** by default), and on node 1, also back up the master database (**IRISDM** by default).
6. 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 next step.
7. On each node to be added to the cluster as a second failover member or DR async, start the ISCAgent, then create the cluster namespace and shard database (**IRISCLUSTER** by default), using the same name and database directory as on the intended first failover member. On the intended second failover member for node 1, as well as any intended DR asyncs for that node, also add the master namespace and database (**IRISDM** by default) in the same manner.
8. If not in a maintenance window, start a new one.
9. 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 creates all the mappings it needs 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.
10. 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 in the *High Availability Guide*; for information using the **iris stop** command to exit the graceful shutdown referred to in that procedure, see Controlling InterSystems IRIS Instances in the *System Administration Guide*.)

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.

4.5.2.6 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 the Mirroring pages of the Management Portal, the ^**MIRROR** routine, or the SYS.MIRROR API), you must update the cluster's metadata by calling **$SYSTEM.Sharding.VerifyShards()** (see %SYSTEM.Sharding API) in the cluster namespace 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, calling **VerifyShards()** 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.

VerifyShards() can be called after every mirroring configuration operation, or can be called once after a sequence of operations, but if operations are performed while the cluster is online, it is advisable to call **VerifyShards()** immediately after any operation which adds or removes a failover member.

Note: Changes made to mirror configurations in a cluster using the %SYSTEM.Cluster API automatically update the cluster metadata and do not require a **VerifyShards()** call.

You can also call **VerifyShards()** to update the cluster's metadata using the **Verify Shards** button on the Sharding Configuration page, as described in the Configure the Shard Master Data Server step of the "Deploying the Namespace-level Architecture" procedure.

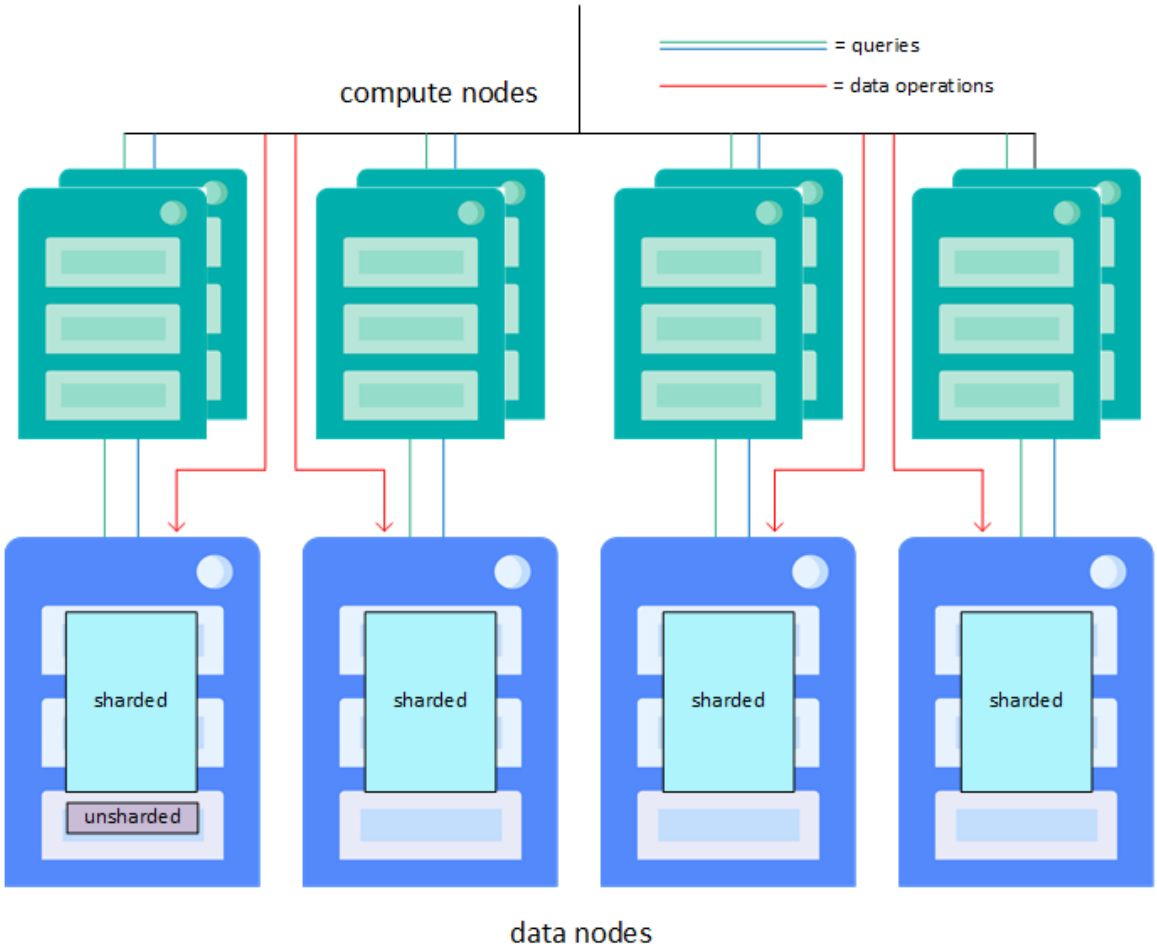
4.5.3 Deploy Compute Nodes for Workload Separation and Increased Query Throughput

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. 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 4–3: 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.

The following sections describe procedures for deploying compute nodes using ICM, the %SYSTEM.Cluster API, and CPF settings. (You also can use the InterSystems Kubernetes Operator IKO) to deploy a sharded cluster with compute nodes on any Kubernetes platform; for details, see Using the InterSystems Kubernetes Operator.)

Note: Before deploying compute nodes using the API or CPF settings, be sure to review both Provision or Identify the Infrastructure and Deploy InterSystems IRIS on the Data Nodes in Deploy the Cluster Using the %SYSTEM.Cluster API for important requirements and recommendations concerning both the infrastructure that hosts the data node mirrors and the InterSystems IRIS instances that are the mirror members.

4.5.3.1 Deploy Compute Nodes Using ICM

To include compute nodes in the sharded cluster when deploying using InterSystems Cloud Manager (ICM), include the desired number of COMPUTE nodes in the definitions file, covered in Define the Deployment. As described in that section, you can use the instance type field to define the compute nodes' total memory and a CPF merge file to determine their database cache size. Compute nodes are automatically assigned to data nodes in round robin fashion, distributing them as evenly as possible. The recommend best practice is to deploy the same number of compute nodes per data node, so define the same number of COMPUTE nodes as DATA nodes, or twice as many, and so on.

To add compute nodes to an existing cluster of data nodes, add a COMPUTE node definition to the definitions.json file and then reprovision and redeploy, as described in Reprovisioning the Infrastructure and Redeploying Services in the "Using ICM" chapter of the *ICM Guide*).

Note: If the number of DATA nodes in the definitions file is greater than the number of COMPUTE nodes, ICM issues a warning.

4.5.3.2 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 in "Deploy the Cluster Using the %SYSTEM.Cluster API"), 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 in *Running InterSystems Products in Containers* and 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 for the instance, see InterSystems IRIS Connection Information in *InterSystems IRIS Basics: Connecting an IDE*.)

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.

4.5.3.3 Configure or Deploy Compute Nodes Using CPF Settings

To configure an existing instance or deploy a new instance as a compute node in a sharded cluster of data nodes, follow the procedure in Configure or deploy the remaining data nodes in "Configure or Deploy the Cluster Using CPF Settings", using the CPF settings listed there, with this modification:

- Set ShardRole to **compute** rather than data.

The following table and CPF excerpt explain and illustrate these settings:

Table 4–7: CPF settings for compute nodes

Section	Setting	Description	Value for remaining data nodes
[Startup]	ShardRole (**CHANGE**)	Determines the node's role in the cluster.	**compute**
	ShardClusterURL	Identifies the existing node to use as a template when adding a node to the cluster, as described in Configure the remaining data nodes in the %SYSTEM.Cluster API procedure. When ShardRole=compute, If a data node is specified, it is used as the template; if a compute node is specified, the data node to which it is assigned is used as the template.	Hostname and superserver port of an existing node in the form IRIS://*host*:*port*.
[config]	MaxServerConn	Sets the maximum number of concurrent connections from ECP clients that an ECP server can accept.	Each of these settings must be equal to or greater than the number of nodes in the cluster. Optionally set them higher than the currently planned number of nodes, to allow for adding nodes later without having to modify them.
	MaxServers	Sets the maximum number of concurrent connections to ECP servers that an ECP client can maintain.	
	globals	Allocates shared memory to the database cache for 8-kilobyte, 16-kilobyte, 32-kilobyte, and 64-kilobyte buffers.	Specify the same target cache size you specified for the data nodes when you created the cluster, as described in Estimate the Database Cache and Database Sizes. For example, to allocate a 200 GB database cache in 8-kilobyte buffers only, the value would be `0,0,204800,0,0,0`.
	gmheap	Optionally configures the size of the generic memory heap.	You probably need to increase this setting from the default of 37.5 MB to optimize the performance of compute nodes. For information about sizing the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

```
[Startup]
ShardRole=compute
ShardClusterURL=IRIS://datanode1:1972
[config]
MaxServerConn=64
MaxServers=64
globals=0,0,204800,0,0,0
gmheap=393,216
```

Configure each compute node instance by updating the indicated settings in its iris.cpf file and restarting it, or deploy the compute node instances using a CPF merge file (as illustrated above) to customize the settings. Once an instance has been deployed or modified and restarted, you can verify that the settings are as desired by viewing the iris.cpf file.

Compute nodes are automatically assigned to data nodes in round robin fashion, distributing them as evenly as possible. The recommended best practice is to deploy the same number of compute nodes per data node, so define the same number of COMPUTE nodes as DATA nodes, or twice as many, and so on.

4.5.4 Install Multiple Data Nodes per System

With a given number of systems hosting data nodes, configuring multiple data node instances per system using the %SYSTEM.Sharding API 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.

4.6 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
- Sharding APIs
- Deploying the Namespace-level Architecture
- Reserved Names

4.6.1 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

4.6.1.1 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.

4.6.1.2 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 4–8: 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 deploying with ICM, you can override the globals setting the configuration parameters file (CPF) by specifying a CPF merge file, as described in Deploying with Customized InterSystems IRIS Configurations in the *ICM Guide*.
  - When deploying using the %SYSTEM.Cluster API, you can use the manual procedure described in Memory and Startup Settings in the *System Administration Guide*.
  - When deploying using CPF settings, you can use the globals CPF setting.

  For guidelines for allocating memory to an InterSystems IRIS instance's routine and database caches as well as the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.
- *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 in the "Configuring InterSystems IRIS" chapter and Maintaining Local Databases in the "Managing InterSystems IRIS" chapter of the *System Administration Guide*.

  When deploying using ICM, you can use the DataVolumeSize parameter (see General Parameters in the *ICM Guide*) to determine 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; this must be large enough to accommodate the target size of the default globals database. On some platforms, this setting is limited by the field DataVolumeType (see Provider-Specific Parameters in the *ICM Guide*).

  Important: When deploying manually, ensure that all instances have database directories and journal directories located on separate storage devices (ICM arranges this automatically). 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 "File System Recommendations" and "Storage Recommendations" in the "Preparing to Install" chapter of the *Installation Guide* and Journaling Best Practices in the "Journaling" chapter of the *Data Integrity Guide*.
- 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 cost tradeoffs 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 (although this must be done with the cluster offline); 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 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 tools like ICM 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 4–9: 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 shard master data server	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 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 tools like ICM can help ensure this consistency.

The recommended best practice is to load balance application connections across all of the data nodes in a cluster. ICM can automatically provision and configure a load balancer for the data nodes as needed when deploying in a public cloud; if deploying a sharded cluster by other means, a load balancing mechanism is required.

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.

4.6.1.3 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 by reprovisioning your ICM deployment 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. (If your definitions file specifies fewer COMPUTE nodes than DATA nodes, ICM issues a warning.)
- 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. 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.

ICM allows you to automatically add a load balancer to your DATA node or COMPUTE node definitions; to load balance across all DATA and COMPUTE nodes, you can provision WS nodes (see ICM Node Types in the "ICM Reference" chapter of the *ICM Guide*), which automatically add all DATA and COMPUTE nodes to their remote server lists. You can also create your own load balancing arrangement.

4.6.2 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 server 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

4.6.2.1 Coordinated Backup and Restore Approaches for Sharded Clusters

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

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 the "Introduction to Data Integrity" chapter of the *Data Integrity Guide*. 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 the failure. For detailed information about journaling, see the "Journaling" chapter of the *Data Integrity Guide*.

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.

4.6.2.2 Coordinated Backup and Restore API Calls

The methods in the Backup.ShardedCluster class can be invoked on a sharded cluster's shard master data server or on one of its shard master application servers (if they exist). All of the methods take a *ShardMasterNamespace* argument; this is the name of either the master namespace on the shard master data server, or the namespace on a shard master application server that is mapped to the default globals database of the master namespace. (For information about how this relationship is configured with the API, see Configure the Shard Master App Servers; ICM creates this configuration automatically, but the result is the same.)

The available methods are as follows:

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

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

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

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

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

  Creates a coordinated journal checkpoint and switches each data server 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 server 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 server 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.

4.6.2.3 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 server 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 server being restored has no mirror failover partner available, and would be used with a mirrored data server only in a disaster recovery situation, as described in Disaster Recovery of Mirrored Sharded Clusters and Disaster Recovery Procedures in the "Mirror Outage Procedures" chapter of the *High Availability Guide*. If the data server being restored is part of a mirror, remove it from the mirror, complete the restore procedure described, and then rebuild it as described in Rebuilding a Mirror Member in the "Managing Mirroring" chapter.

Create Coordinated Backups

1. Call **Backup.ShardedCluster.Quiesce**, which pauses activity on all data servers 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 servers in the cluster. Although the database backups are coordinated, they may include open transactions; when the data servers 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 server 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 server:

   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 server to be inconsistent with the others.

c. Restart the data server.

The data server 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 servers in the cluster, create cold backups, and restart the data servers.

Create Uncoordinated Backups Followed by Coordinated Journal Checkpoints

1. Create backups of the databases on all data servers in the cluster while the data servers 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 server 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 servers can be recovered, rather than the timing of the backups.

   Note: Before switching journal files, **JournalCheckpoint()** briefly quiesces all data servers 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 server, 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 servers 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 servers. This requires means that for each data server 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 server to be inconsistent with the others.

5. On each data server, 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 server 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 servers 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 server 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.

2. Create backups of all data servers in the cluster. The databases backups are not coordinated and may also include partial transactions, but when restoring the data servers 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 server 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 server:

   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 in the "Backup and Restore" chapter of the *Data Integrity Guide* 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 endi 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 in *Running InterSystems Products in Containers* for instructions for doing a manual recovery inside a container.

   The data server 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 server be located such that a single backup can include them both.

4.6.3 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 in the *High Availability Guide*.

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 in the *High Availability Guide*.
   - 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.

4.6.4 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).

4.6.4.1 %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:

- **$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.

4.6.4.2 %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:

- **$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()**

  When converting to nonmirrored cluster to a mirrored cluster, adds the master and shard databases of data nodes that have been added as mirror 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.Help()**

  Displays a summary of the methods of %SYSTEM.Sharding.

4.6.5 Deploying the Namespace-level Architecture

Use the following procedure to deploy an InterSystems IRIS sharded cluster with the older namespace-level architecture, consisting of a shard master, shard data servers, and optionally shard master application servers using the %SYSTEM.Sharding API. Instructions are also provided for deploying the cluster using the Sharding Configuration page in the Management Portal (**System Administration > Configuration > System Configuration > Sharding Configuration**).

Note: As with all classes in the %SYSTEM package, the %SYSTEM.Sharding methods are available through $SYSTEM.Sharding.

This procedure assumes each InterSystems IRIS instance is installed on its own system. Use the following procedures to deploy a namespace-level cluster.

- Provision or identify the infrastructure
- Install InterSystems IRIS on the cluster nodes
- Configure the cluster nodes
  - Configure IP addresses for cluster communications (optional)
  - Configure the shard data servers
  - Configure the shard master data server
  - Configure the shard master app servers

4.6.5.1 Provision or Identify the Infrastructure

Identify the needed number of networked host systems (physical, virtual, or cloud) — one host each for the shard master, shard data servers, and shard master app servers (if any).

Important: Be sure to review provision or identify the infrastructure in Deploy Cluster Using the %SYSTEM.Cluster API for requirements and best practices for the infrastructure of a sharded cluster.

4.6.5.2 Install 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 in Deploy Cluster Using the %SYSTEM.Cluster API 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 "File System Recommendations" and "Storage Recommendations" in the "Preparing to Install" chapter of the *Installation Guide* and Journaling Best Practices in the "Journaling" chapter of the *Data Integrity Guide*.

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 in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.

   Note: In some cases, it may be advisable to increase the size of the generic memory heap on the cluster members. For information on how to allocate memory to the generic memory heap, see gmheap in the *Configuration Parameter File Reference*.

   For guidelines for allocating memory to an InterSystems IRIS instance's routine and database caches as well as the generic memory heap, see Calculating Memory Requirements and Allocation in the "Vertically Scaling InterSystems IRIS" chapter.

4.6.5.3 Configure the Cluster Nodes

Perform the following steps on the instances with each role in the cluster.

Configure IP Addresses for Cluster Communications (Optional)

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.

Configure the Shard Data Servers

On each shard data server instance:

1. Create the shard namespace using the Management Portal, as described in Create/Modify a Namespace in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*. (The namespace need not be interoperability-enabled.)

   Create a new database for 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.

   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 in *Running InterSystems Products in Containers* and 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 for the instance, see InterSystems IRIS Connection Information in *InterSystems IRIS Basics: Connecting an IDE*.)

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, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.

Management Portal

Take the following steps to deploy using the Management Portal instead of the API:

- Create the shard namespace by following the instructions in step 1, and make sure you have recorded the needed information about the instance as detailed in step 2.
- Navigate to the Sharding Configuration page (**System Administration > Configuration > System Configuration > Sharding Configuration**) and use the **Enable Sharding** button to enable sharding. Then restart the instance, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.

Configure the Shard Master Data Server

On the shard master data server instance:

1. Create the master namespace using the Management Portal, as described in Create/Modify a Namespace in the "Configuring InterSystems IRIS" chapter of the *Administration Guide*. (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.

   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, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.

   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")
   ```

   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 of the procedure 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.

After configuring shard master application servers as described in the next section, you can call **$SYSTEM.Sharding.VerifyShards()** on each of them as well to confirm that they can communicate with the shard master data server and the shards.

Management Portal:

Take the following steps to deploy using the Management Portal instead of the API:

- Create the master namespace by following the instructions in step 1.
- Navigate to the Sharding Configuration page (**System Administration > Configuration > System Configuration > Sharding Configuration**) and use the **Enable Sharding** button to enable sharding, Then restart the instance, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.
- Return to the Sharding Configuration page (reloading if necessary) and for each shard data server,
  - Click the **Assign Shard** button and enter the shard data server's host, the instance's superserver port, and the name of the shard namespace in the **Assign Shard** dialog. Leave the drop-down set to **Data Shard**, and leave the **Mirrored** checkbox cleared. Click **Finish** to assign the shard data server to the cluster.
  - Click the **Verify Shards** button to verify that the shards have been correctly configured and that the shard master can communicate with them. If the operation reports an error, you can use the **Edit** link to review and if necessary correct the information you entered, or the **Deassign** link to deassign the shard data server and repeat the **Assign Shard** operation.

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

Configure the Shard Master App Servers

On each shard master app server (if you are configuring them):

1. In a 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. After making this call, restart the instance, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.

2. As described in Configuring an Application Server in the "Horizontally Scaling Systems for User Volume with InterSystems Distributed Caching" chapter of this guide:
   - Add the shard master data server as a data server.

     Note: Do not change the Maximum number of data servers and Maximum number of application servers settings on the ECP Settings page, which were specified by the **$SYSTEM.Sharding.EnableSharding()** call.

   - Create a namespace on the shard master data server and configure the default globals and routines databases of the master namespace on the shard master data server as the default globals and routines databases of the namespace on the shard master app server, thereby adding them as remote databases. This will be the namespace in which to execute queries, rather than the master namespace on the shard master data server.

If you have configured shard master app servers, configure the desired mechanism to distribute application connections across them.

Management Portal:

Take the following steps to deploy using the Management Portal instead of the API:

- Navigate to the Sharding Configuration page (**System Administration > Configuration > System Configuration > Sharding Configuration**) and use the **Enable Sharding** button to enable sharding. Then restart the instance, unless you had previously changed the values of the **MaxServerConn** and **MaxServers** CPF settings as described in Deploy InterSystems IRIS on the Data Nodes in the procedure for deploying a sharded cluster using the %SYSTEM.Cluster API.
- Add the shard master data server as a data server, create a namespace, and configure the master namespace's globals and routines databases as the databases for the new namespace, as described in step 2.

4.6.6 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 classnames 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.