First Look: Deploying an InterSystems Sharded Cluster

Version 2018.1
2018-10-22

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

First Look: Deploying an InterSystems Sharded Cluster
InterSystems IRIS Data Platform Version 2018.1 2018-10-22

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

First Look: Deploying an InterSystems Sharded Cluster

This First Look guide introduces you to the InterSystems IRIS™ sharding feature and its use in a sharded cluster to horizontally scale InterSystems IRIS Data Platform™ for data volume.

As part of this guide, you will use ICM to provision a sharded cluster in a public cloud and see how sharding a table distributes its rows across the shards in a cluster.

1 How Can Sharding Help You?

Are you feeling the Big Data heat?

Ready or not, we are all managing more data than ever before and being asked to do more with it — and the response times demanded are growing ever shorter. 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 data workload but *scale* to meet increasing demand while maintaining the performance standards you rely on and avoiding business disruption. Each business-specific workload presents different challenges to the data platform on which it operates — and as workloads grow, those challenges become even more acute.

InterSystems IRIS includes a comprehensive set of capabilities to scale your applications, which can be applied alone or in combination, depending on the nature of your workload and the specific performance challenges it faces. One of these, sharding, partitions both data and its associated cache across a number of servers, providing flexible, inexpensive performance scaling for queries and data ingestion while maximizing infrastructure value through highly efficient resource utilization. An InterSystems IRIS *sharded cluster* can provide significant performance benefits for a wide variety of applications, but especially for those with workloads that include one or more of the following:

- High-volume or high-speed data ingestion, or a combination.
- Relatively large data sets, queries that return large amounts of data, or both.
- Complex queries that do large amounts of data processing, such as those that scan a lot of data on disk or involve significant compute work.

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 all three factors — 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.

Note that these characteristics all have to do with data; the primary function of InterSystems IRIS sharding is to *scale for data volume*. But a sharded cluster can also include features that *scale for user volume*, when workloads involving some or all of these data-related factors also experience a very high query volume from large numbers of users. And sharding can be combined with vertical scaling as well. With InterSystems IRIS, you can create just the right overall scaling solution for your workload's performance challenges.

2 How Does Sharding Work?

The heart of the sharded architecture is the partitioning of data and its associated cache across a number of systems. A sharded cluster partitions large database tables horizontally — that is, by row — across multiple InterSystems IRIS instances, called *shard data servers*, while allowing applications to transparently access these tables through a single instance, called the *shard master data server* (or just shard master). This architecture provides three advantages:

- Parallel processing

  Queries directed to the shard master are run in parallel on the shard data servers, with the results merged, combined, and returned to the application as full query results by the shard master, significantly enhancing execution speed in many cases.

- Partitioned caching

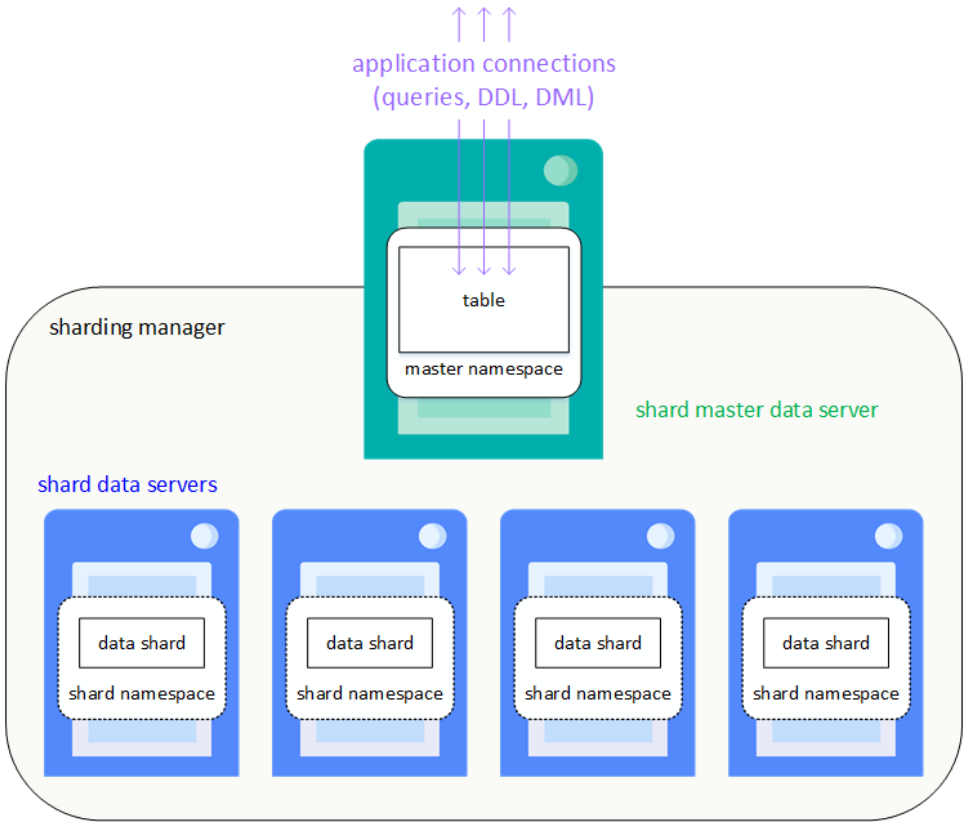
  Each shard data server has its own dedicated cache, rather than a single instance's cache serving the entire data set, which greatly reduces the risk of overflowing the cache and forcing performance-degrading disk reads.

- Parallel loading

  Data can be loaded onto the shards in parallel, reducing cache and disk contention between the ingestion workload and the query workload and improving the performance of both.

Nonsharded tables are stored on the shard master, while a federated software component called the *sharding manager* keeps track of which data is on which shard data servers and directs queries accordingly. From the perspective of the application SQL, the distinction between sharded and nonsharded tables is totally transparent.

Figure 1: A Basic Sharded Cluster



3 Try It! Deploy and Demo an InterSystems IRIS Sharded Cluster

In this exploration, you will

- Deploy a basic sharded cluster in a public cloud using InterSystems Cloud Manager (ICM).

  ICM is an automated command-line tool that makes it easy to provision infrastructure in the cloud and deploy InterSystems IRIS and other services on the provisioned nodes. ICM takes care of all the needed InterSystems IRIS configuration, so if you specify a sharded cluster, the cluster is ready to use when deployment is complete.

- Create three sharded tables from the same data using different shard keys and observe the different distributions of the rows across the shards.

  The shard key is the field or fields used to determine which rows of a sharded table are stored on which shards. By default, the RowID is used as the shard key, and this distribution is the most effective approach for most sharded tables because it offers the best guarantee of an even distribution of data and allows the most efficient parallel data loading. There is one case, however, in which a user-defined key can be beneficial. When you shard two large tables that are frequently joined in queries on the field or fields used to join them, the rows to be joined are stored on the same shards, and a join can be executed locally on each shard rather than across the shards, providing a significant performance boost. This is called a *cosharded join*.

- Run the same query on all three sharded tables to demonstrate that the distribution of rows across shards is entirely transparent to the query.

To introduce you to sharding without bogging you down in details, we've kept this simple. A sharded cluster in production requires planning and a number of decisions, so be sure not to confuse this exploration of sharding with the real thing! For example, when designing a production cluster, you would review your schema and tables to decide which tables are to be sharded, then decide how many shard data servers to deploy (typically on the order of 4 to 16) based on both the anticipated working set for the sharded data and the amount of memory available for the database caches on your servers. In this exploration, however, you'll deploy a basic two-shard cluster. The sources listed at the end of this document will give you a good idea of what's involved in using sharding in production. The chapter "Horizontally Scaling InterSystems IRIS for Data Volume with Sharding" in the *Scalability Guide* provides detailed information about sharding and sharded clusters.

These instructions assume you have an InterSystems IRIS sharding license and access to InterSystems software downloads.

3.1 Use ICM to Deploy the Cluster

You can find the procedure for using ICM to deploy the sharded cluster on the Amazon Web Services public cloud platform in *First Look: InterSystems Cloud Manager*; specifically in the section Try It! Deploy InterSystems IRIS in the Cloud with ICM. You can use the entire procedure as it is presented there. When you get to Customize definitions.json, be sure to make only one change to the definitions.json file — replace the value of the ISCLicense field with the location of your InterSystems IRIS sharding license within the ICM container. When you have finished the deployment stage (Deploy InterSystems IRIS), remain in the ICM container.

Note: As an alternative, if you prefer, you can install the InterSystems IRIS instances on existing physical, virtual or cloud machines and deploy the sharded cluster using the Sharding API, as described in Deploy the Cluster Using the Sharding API in the "Horizontally Scaling InterSystems IRIS for Data Volume with Sharding" chapter of the *Scalability Guide*.

3.2 Use Different Shard Keys to Distribute Rows Differently, Then Query the Sharded Tables

To see how a sharded cluster partitions a sharded table across the shards based on the shard key you use, you are going to connect to the shard master and take the following steps:

- Create and populate a small nonsharded table.
- Create three sharded tables with the same fields as the nonsharded table, using three different shard keys.
- Populate the sharded tables from the nonsharded table.
- Select the rows from the sharded tables to see how the rows are distributed differently across the shards in each case.

  When a table is sharded, RowID values are assigned from different ranges for each shard, starting at widely-separated ranges; rows in the same range are on the same shard. When dealing with a small table, as in this case, the shard RowIDs are easy to distinguish, clearly showing you which rows are on one shard and which on the other (although you cannot tell which shard is which). Rows of tables sharded on RowID are distributed in a way that has no relationship with the data in the rows, and which would likely be different if you emptied and reloaded a table. A user-defined shard key, on the other hand, distributes rows based on the values of the field or fields in the key, and will therefore be the same when you reload the table (assuming the values and the number of shards have not changed).
- Query all three tables for the maximum length of one of the fields to show that the distribution of rows across shards is transparent to the query.

Use the procedure that follows for this part of the exploration:

1. Open a Terminal window on the shard master instance.
   - On the ICM command line, issue the command

     ```
     icm terminal -machine <role>-DM-<tag>-0001
     ```

     where *role* and *tag* are the values you inserted into the defaults.json file as the values for the Role and Tag fields. If you are uncertain about the name of the shard master node, you can use the **icm inventory** command to display the node names:

     ```
     $ icm inventory
     Machine             IP Address       DNS Name
     -------             ----------       --------
     MCG-DM-TEST-0001    00.53.183.209    ec2-52-53-183-209.us-west-1.compute.amazonaws.com
     MCG-DS-TEST-0001    00.53.183.185    ec2-52-53-183-185.us-west-1.compute.amazonaws.com
     MCG-DS-TEST-0002    00.56.59.42      ec2-13-56-59-42.us-west-1.compute.amazonaws.com
     ```

     The **icm terminal** command opens a Terminal window for the InterSystems IRIS instance on the node you specify, in this case the shard master data server.

   - If you deployed your cluster manually using the Sharding API, open a Terminal window on the shard master instance.

2. In the Terminal window, switch to the cluster's master namespace.
   - In the defaults.json file you customized for ICM, this is **DB**, so if you did not change this field, switch to the DB namespace:

     ```
     Node: MCG-DM-TEST-0001, Instance: IRIS

     USER> zn "DB"

     DB>
     ```

     Otherwise, switch to the namespace you specified in defaults.json.

- If you deployed your cluster manually using the Sharding API, change to the master namespace you created when configuring the shard master data server.

3. Open the Terminal SQL shell:

```
DB> do $SYSTEM.SQL.Shell()

[SQL]DB>>
```

4. Create and populate the nonsharded table, test.nonsharded, with the following SQL statements.

```
CREATE TABLE test.nonsharded (field1 CHAR(5), field2 CHAR(5))
INSERT INTO test.nonsharded (field1,field2) VALUES ('one','one')
INSERT INTO test.nonsharded (field1,field2) VALUES ('one','two')
INSERT INTO test.nonsharded (field1,field2) VALUES ('one','three')
INSERT INTO test.nonsharded (field1,field2) VALUES ('two','one')
INSERT INTO test.nonsharded (field1,field2) VALUES ('two','two')
INSERT INTO test.nonsharded (field1,field2) VALUES ('two','three')
INSERT INTO test.nonsharded (field1,field2) VALUES ('three','one')
INSERT INTO test.nonsharded (field1,field2) VALUES ('three','two')
INSERT INTO test.nonsharded (field1,field2) VALUES ('three','three')
```

   Use SELECT to see the contents of the table:

```
SELECT * FROM test.nonsharded

field1      field2
one     one
one     two
one     three
two     one
two     two
two     three
three   one
three   two
three   three
```

5. Create three sharded tables with the same fields that are in test.nonsharded, using the default shard key (RowID) for the first one, the **field1** field for the second one, and the **field2** field for the third, and populate each using an INSERT INTO statement selecting from test.nonsharded:

```
CREATE TABLE test.rowid (field1 CHAR(5), field2 CHAR(5), SHARD)
INSERT INTO test.rowid (field1,field2) SELECT field1,field2 FROM test.nonsharded

CREATE TABLE test.field1 (field1 CHAR(5), field2 CHAR(5), SHARD KEY (field1))
INSERT INTO test.field1 (field1,field2) SELECT field1,field2 FROM test.nonsharded

CREATE TABLE test.field2 (field1 CHAR(5), field2 CHAR(5), SHARD KEY (field2))
INSERT INTO test.field2 (field1,field2) SELECT field1,field2 FROM test.nonsharded
```

6. Use SELECT *,%ID to display the contents of each sharded table with its RowID on the shard.

```
SELECT *,%ID FROM test.rowid ORDER BY %ID

field1      field2      ID
one     one     1
one     two     2
two     one     3
two     two     4
three   three   5
one     three   256000
two     three   256001
three   one     256002
three   two     256003
```

   This distribution does not reflect the values of field1 or field2 (rows with all three values of each are located on both shards). If you delete, recreate, and reload test.rowid, the distribution may be different.

```
SELECT *,%ID FROM test.field1 ORDER BY %ID
```

```
field1      field2      ID
one     one     1
one     two     2
one     three   3
two     one     256000
two     two     256001
two     three   256002
three   one     256003
three   two     256004
three   three   256005
```

Sharding on the **field1** field distributes the rows so that those with the same value of **field1** are placed on the same shard. In this case, rows with value **one** are on one shard and those with values **two** and **three** on the other, but which values end up on which shard depends on how many shards and how many values there are.

```
SELECT *,%ID FROM test.field2 ORDER BY %ID
```

```
field1      field2      ID
one     one     1
two     one     2
three   one     3
one     two     256000
one     three   256001
two     two     256002
two     three   256003
three   two     256004
three   three   256005
```

Here, distribution is determined by the value of the **field2** field.

7. Finally, as an example of how sharding distributes work to the shard data servers, use the following SELECT statement with all three sharded tables:

```
SELECT MAX(LENGTH(field2)) FROM <table>
```

In each case, the result is **5**, the length of the longest value, **three**, because the distribution of rows across the shards is entirely transparent to the query. The MAX(LENGTH(field2)) expression is computed independently on each shard, and the shard master chooses the MAX() of the results they return. For example, when the query is run on the test.field2 table, one shard returns **3**, because it has only the value **one** in the **field2** field, while the other shard returns **5**; the shard master then chooses **5** as the larger of the two results.

If you like, use **EXPLAIN** to show the query plan indicating explicitly how work is sent to the shards:

```
EXPLAIN SELECT MAX(LENGTH(field2)) FROM <table>
```

4 More Sharded Cluster Options

Additional options for a sharded cluster include:

- You can horizontally scale a sharded cluster for user volume by distributing both application logic and caching across a tier of *shard master app servers* sitting in front of the shard master data sever. User connections can be distributed across the tier 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. (Elements of Sharding in the *Scalability Guide*)
- *Query shards* (hosted by shard query servers) provide query access to the data stored on the shard data servers. To minimize interference between the query and data ingestion workloads, you can deploy a single query shard per shard data server; to increase the throughput of a sharded cluster handling a high volume multiuser workload, deploy multiple query shards per shard data server. (Deploy Query Shards in the *Scalability Guide*)

- To add high availability for the data on the cluster, you can deploy both the shard master data server (nonsharded tables) and shard data servers (sharded tables) as mirrored failover pairs. (Mirror for High Availability in the *Scalability Guide*.)

5 Learn More About Sharding

To learn more about sharding, see

- Data Platform Scalability Technology Overview (video)
- Introduction to Sharding (online course)
- Sharding Basics (online course)
- InterSystems in the Cloud Experience (online experience)
- Scalability Guide
- First Look: InterSystems Cloud Manager