InterSystems™
IRIS Data Platform

Using the InterSystems Kubernetes Operator

Version 2020.3
2021-02-04

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

Table of Contents

Using the InterSystems Kubernetes Operator

This document explains how to use the InterSystems Kubernetes Operator (IKO) to deploy InterSystems IRIS sharded clusters on Kubernetes platforms.

1 Why would I use Kubernetes?

Kubernetes is an open-source orchestration engine for automating deployment, scaling, and management of containerized workloads and services, and excels at orchestrating complex SaaS (software as a service) applications. You provision a Kubernetes-enabled cluster and tell Kubernetes the containerized services you want to deploy on it and the policies you want them to be governed by; Kubernetes transparently provides the needed resources in the most efficient way possible, repairs or restores the configuration when problems with those resources cause it to deviate from what you specified, and can scale automatically or on demand. In the simplest terms, Kubernetes deploys a multicontainer application in the configuration and at the scale you specify on any Kubernetes-enabled platform, and keeps the application operating exactly as you described it.

2 Why do I need the InterSystems Kubernetes Operator?

The InterSystems Kubernetes Operator (IKO) extends the Kubernetes API with a custom resource representing an InterSystems IRIS sharded cluster. This resource, called an *iriscluster*, can be deployed on any Kubernetes platform on which the IKO is installed. (The IKO can also deploy a stand-alone instance of InterSystems IRIS on Kubernetes.)

The operator isn't required to deploy InterSystems IRIS under Kubernetes. But because Kubernetes is application-independent, you would need to create custom definitions and scripts to handle all the needed configuration of the InterSystems IRIS instances or other components in the deployed containers along with networking, persistent storage requirements, and so on. Installing the IKO automates these tasks. By putting together a few settings that define the cluster, for example the number of data and compute nodes, whether they should be mirrored, and where the Docker credentials needed to pull the container images are stored, you can easily deploy your InterSystems IRIS cluster exactly as you want it. The operator also adds InterSystems IRIS-specific cluster management capabilities to Kubernetes, enabling tasks like adding data or compute nodes, which you would otherwise have to do manually by interacting directly with the instances.

3 Start with your use case

Before beginning your work with the IKO, you should confirm that:

- The InterSystems IRIS sharding architecture will be beneficial for the anticipated workload.
- Containerized deployment of InterSystems IRIS and your application is the best approach.
- Kubernetes is the right orchestrator for your containerized InterSystems-IRIS based application, and that you have identified one or more suitable Kubernetes platforms to deploy it on. For example, major public clouds platforms

include Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), Amazon Elastic Container Service for Kubernetes (EKS), and Tencent Kubernetes Engine (TKE), while platforms such as Red Hat OpenShift, Rancher Kubernetes Engine (RKE), and Docker Enterprise can be used on any infrastructure.

4 Plan your sharded cluster

For the most beneficial results, it is important to fully plan the configuration of your sharded cluster and its data, including:

- The number of data nodes in the cluster and their configuration, such as their database cache size, the storage used for their default databases, and so on)
- Whether the data nodes are to mirrored for high availability
- Whether to include compute nodes for workload separation and increased query throughput
- The schema for the sharded and nonsharded data to be loaded onto the cluster.

For detailed information about InterSystems IRIS sharded clusters, see Horizontally Scaling for Data Volume with Sharding in the *Scalability Guide*.

5 Learn to speak Kubernetes

While it is possible to use the IKO if you have not already worked with Kubernetes, InterSystems recommends having or gaining a working familiarity with Kubernetes before deploying with the IKO.

6 Choose a platform and understand the interface

When you have selected the Kubernetes platform you will deploy on, create an account and familiarize yourself with the provided interface(s) to Kubernetes. For example, to use GKE on Google Cloud Platform, you can open a Google Cloud Shell terminal and file editor to use GCP's **gcloud** command line interface and the Kubernetes **kubectl** command line interface. Bear in mind that the configuration of your Kubernetes environment should include access to the availability zones in which you want to deploy the sharded cluster.

The instructions in this document provide examples of **gcloud** commands

7 Deploy a Kubernetes container cluster to host the iriscluster

The Kubernetes cluster is the structure on which your containerized services are deployed and through which they are scaled and managed. The procedure for deploying a cluster varies to some degree among platforms. In planning and deploying your Kubernetes cluster, bear in mind the following considerations:

- The IKO deploys one InterSystems IRIS or arbiter container (if a mirrored cluster) per Kubernetes pod, and attempts to deploy one pod per Kubernetes cluster node when possible. Ensure that

  - You are deploying the desired number of nodes to host the pods of your sharded cluster, including the needed distribution across zones if more than one zone is specified (see below).
  - The required compute and storage resources will be available to those nodes.

- If your sharded cluster is to be mirrored and you plan to enforce zone antiaffinity using the `preferredZones` fields in the iriscluster definition to deploy the members of each failover pair in separate zones and the arbiter in an additional zone, the container cluster must be deployed in three zones. For example, if you plan to use zone antiaffinity and are deploying the cluster using the *gcloud* command-line interface, you might select zones *us-east1-b,c,d* and create the container cluster with a command like this:

  ```
  $ gcloud container clusters create my-iriscluster --node-locations us-east1-b,us-east1-c,us-east1-d
  ```

8 Upgrade Helm if necessary

Helm packages Kubernetes applications as *charts*, making it easy to install them on any Kubernetes platform. Because the IKO Helm chart requires Helm version 3, you must confirm that this is the version on your platform, which you can do by issuing the command **helm version**. If you need to upgrade Helm to version 3, you can use the curl script at https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3. For example:

```
$ curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6827  100  6827    0     0  74998      0 --:--:-- --:--:-- --:--:-- 75855
Helm v3.2.3 is available. Changing from version .
Downloading https://get.helm.sh/helm-v3.2.3-linux-amd64.tar.gz
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

9 Download the IKO archive and upload the extracted contents to Kubernetes

Obtain the IKO archive file, for example iris_operator-2.0.0.223.0-unix.tar.gz, from the InterSystems Worldwide Response Center (WRC) download area and extract its contents. Next, upload the extracted directory, iris_operator-*version* (for example iris_operator-2.0.0.223.0) to the Kubernetes platform. This directory contains the following:

- The image/ directory contains an archive file containing the IKO image.
- The chart/iris-operator directory contains the Helm chart for the operator.
- The samples/ directory contains template .yaml and .cpf files, as described later in this procedure.

10 Locate the IKO image

To install the IKO, Kubernetes must be able to download (pull) the IKO image. To enable this, you must provide Kubernetes with the registry, repository, and tag of the IKO image and the credentials it will use to authenticate to the registry. Generally, there are two approaches to downloading the image:

- The IKO image is available from the InterSystems Container Registry (ICR). Using the InterSystems Container Registry lists the images currently available from the ICR, for example **containers.intersystems.com/iris-operator:2.0.0.223.0**, and explains how to obtain login credentials for the ICR.
- You can use Docker commands to load the image from the image archive you extracted from the IKO archive in the previous step, then add it to the appropriate repository in your organization's container registry, for example:

```
$ docker load -i iris_operator-2.0.0.223.0/image/iris_operator-2.0.0.223.0-docker.tgz
fd6fa224ea91: Loading layer [==================================================>] 3.031MB/3.031MB

32bd42e80893: Loading layer [==================================================>] 75.55MB/75.55MB

Loaded image: intersystems/iris-operator:2.0.0.223.0
$ docker images
REPOSITORY                  TAG          IMAGE ID      CREATED       SIZE
intersystems/iris-operator  2.0.0.223.0  9a3756aed423  3 months ago  77.3MB
$ docker tag intersystems/iris-operator:2.0.0.223.0 kubernetes/intersystems-operator
$ docker login docker.acme.com
Username: pmartinez@acme.com
Pasword: **********
Login Succeeded
$ docker push kubernetes/intersystems-operator
The push refers to repository [docker.acme.com/kubernetes/intersystems-operator]
4393194860cb: Pushed
0011f6346dc8: Pushed
340dc52ed535: Pushed
latest: sha256:f483e14a1c6b7a13bb7ec0ab1c69f4588da2c253e8765232 size 77320
```

11 Create a secret for IKO image pull information

Kubernetes secrets let you securely and flexibly store and manage sensitive information such as credentials that you want to pass to Kubernetes. When you want Kubernetes download an image, you can create a Kubernetes secret of type **docker-registry** containing the URL of the registry and the credentials needed to log into that registry to pull the images from it. Create such a secret for the IKO image you located in the previous step. For example, if you pushed the image to your own registry, you would use a **kubectl** command like the following to create the needed secret. The username and password in this case would be your credentials for authenticating to the registry (docker-email is optional).

```
$ kubectl create secret docker-registry acme-pull-secret
  --docker-server=https://docker.acme.com --docker-username=*****
  --docker-password='*****' --docker-email=**********
```

12 Update the values.yaml file

In the chart/iris-operator directory, ensure that the fields in operator section near the top of the values.yaml file correctly describe the IKO image you want to pull to install the IKO, for example:

```
operator:
  registry: docker.acme.com/kubernetes
  repository: intersystems-operator
  tag: latest
```

Further down in the file, in the imagePullSecrets section, provide the name of the secret you created to hold the credentials for this registry, for example:

```
imagePullSecrets:
  name: acme-pull-secret
```

13 Install the IKO

Use Helm to install the operator on the Kubernetes cluster. For example, on GKE you would use the following command:

```
$ helm install intersystems iris_operator-2.0.0.323.0/chart/iris-operator
NAME: intersystems
LAST DEPLOYED: Mon Jun 15 16:43:21 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
To verify that InterSystems Kubernetes Operator has started, run:
  kubectl --namespace=default get deployments -l "release=intersystems, app=iris-operator"
bbinstoc@cloudshell:~ (isc-mktplace-dev)$ kubectl get deployments -l "release=intersystems,
app=iris-operator"
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
intersystems-iris-operator   0/1     1            0           30s
```

14 Choose the default InterSystems IRIS password and generate a hash

InterSystems IRIS is installed with several predefined user accounts, the initial password for which is **SYS**. For effective security, it is important that this default password be changed immediately upon deployment of your container. For this reason, the IKO requires you to include a cryptographic hash (with salt) of the new default password in the YAML definition of your iriscluster. InterSystems publishes through the ICR the **intersystems/passwordhash** image, from which you can create a container that generates the hash for you; for more information, see Authentication and Passwords in *Running InterSystems Products in Cotainers*.

15 Create a secret for InterSystems IRIS image pull information

Create another Kubernetes secret, like the one you created for the IKO image pull information, for the InterSystems IRIS image and (if the cluster is to be mirrored) the arbiter image to be deployed. For example, if Kubernetes will be pulling the image from the InterSystems Container Registry (ICR) as previously described,, you would use a command like the following. The username and password in this case would be your ICR docker credentials, which you can obtain as described in Authenticating to the ICR in *Using the InterSystems Container Registry* (`docker-email` is optional).

```
$ kubectl create secret docker-registry intersystems-pull-secret
  --docker-server=https://containers.intersystems.com --docker-username=*****
  --docker-password='*****' --docker-email=**********
```

16 Create a secret containing the InterSystems IRIS license key

Upload the sharding-enabled license key for the InterSystems IRIS images in your sharded cluster, and create a Kubernetes secret of type **`generic`** to contain the key, allowing it to be mounted on a temporary file system within the container, for example:

```
$ kubectl create secret generic iris-key-secret --from-file=iris.key
```

17 Create configuration parameter files and a config map for them

The configuration parameter file, also called the CPF, defines the configuration of an InterSystems IRIS instance. On startup, InterSystems IRIS reads the CPF to obtain the values for most of its settings. The CPF merge feature allows you to specify a merge file containing configuration that are different from, and overwrite, the settings in the default CPF that comes with an instance when it is deployed. For details, see Introduction to the Configuration Parameter File.

Kubernetes config maps keep your containerized applications portable by separating configuration artifacts, such as CPF merge files, from image content. If you want to use separate CPF merge files to update the configurations of the iriscluster's data nodes and compute nodes at deployment, you must do two things:

- Customize the template *data.cpf* and *compute.cpf* files provided in the samples/ directory with the CPF settings you want to apply to the data and compute nodes. (The templates contain only the SystemMode setting, which displays text on the InterSystems IRIS management portal.) For example, as described in Deploying the Sharded Cluster in the *Scalability Guide*, the data nodes in a sharded cluster must be configured to allocate a database cache of the appropriate size, and typically the default generic memory heap size should be increased as well. To do this, you would add the **[config]** section globals and gmheap parameters to the *data.cpf* file, as shown below, to configure the database cache and generic memory heap of each data node at 200 GB and 256 MB, respectively:

  ```
  [StartUp]
  SystemMode=my-iriscluster
  [config]
  globals= 0,0,204800,0,0,0
  gmheap= 262144
  ```

- Create a Kubernetes config map specifying the files, using a command like this:

  ```
  $ kubectl create cm iris-cpf --from-file data.cpf --from-file compute.cpf
  ```

18 Create a storage class for persistent storage

Kubernetes provides the persistent storage needed by containerized programs (in this case, the data nodes) in the form of persistent volumes; a persistent volume claim (pvc) is a request by a user for a persistent volume. To include these in the cluster, you need to specify in the definition a Kubernetes storage class, which determines the type of storage used. Because storage types are platform specific, the storage class must be defined separately from the cluster, which is why multiple template storage class definition files are provided with the IKO. For example, in the samples/ directory, the template storage class definition file for GKE is named iris-ssd-sc-gke.yaml and has the following contents:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: iris-ssd-storageclass
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
volumeBindingMode: WaitForFirstConsumer
```

The `volumeBindingMode: WaitForFirstConsumer` setting is required for the correct operation of the IKO.

Update the storage class definition as needed and then create the storage class, for example:

```
$ kubectl create -f iris-ssd-sc-gke.yaml
```

Using the definition file shown above, this command creates a storage class called **`iris-ssd-storageclass`**.

19 Create the iriscluster definition file

You are now ready to create your iriscluster definition file by customizing the template file provided with the IKO in the samples/ directory, my-iriscluster-definition.yaml. The contents of this file and suggestions for customizing it are shown in the following sections.

Note: For each section of the definition, there are numerous other Kubernetes fields that can be included; this section discusses only those specific to or required for an iriscluster definition.

```
apiVersion: intersystems.com/v1alpha1
kind: IrisCluster
metadata:
  name: my-iriscluster
spec:
```

The first three fields are required by Kubernetes, as is the `spec:` section, which contains nested fields specifying a particular type of object, in this case an iriscluster. Change the value of the `name:`field in `metadata:` to the name you want to give the cluster.

```
  passwordHash: ''
```

The `passwordHash` field specifies the cryptographic hash of the desired default password for the InterSystems IRIS instances in the cluster; insert the hash you generated as the value.

```
  licenseKeySecret:
    name: iris-key-secret
```

The `licenseKeySecret` is the Kubernetes secret containing the InterSystems IRIS license key; update the `name:` field with the name of the license key secret you created.

```
  configSource:
    name: iris-cpf
```

The `configSource` is the config map containing the CPF merge files, data.cpf and compute.cpf. Update the `name:` field with the name of the config map you created; if you did not create one, do not specify a value for this field.

```
  topology:
```

The topology section specifies the details of the cluster’s data nodes, compute nodes (if any), and arbiter node (if mirrored). Only the `data:` section is required.

```
    data:
      shards: <1..N>
```

In the `data:` definition, the `shards:` field specifies the number of data nodes to deploy. Data nodes can be added to the deployed cluster by increasing this setting and reapplying the definition, but the setting cannot be decreased.

Note: The `shards:` field is optional, with a default value of 1. If it is omitted, and the `compute:` and `arbiter:` sections of the definition are also omitted, applying the definition results in stand-alone deployment of a single instance of InterSystems IRIS, rather than a sharded cluster. If the `mirrored:` field, described below, is set to `true`, two stand-alone instances are deployed as the primary and backup in a mirror (and you must include the `arbiter:` section, described later, in the definition).

```
mirrored: <true|false>
```

If `mirrored: true`, data nodes are mirrored, and two instances are deployed for each data node specified by the `shards:` field. For example, if `shards: 4` and `mirrored: true`, eight data node instances are deployed as four failover pairs.

```
image: containers.intersystems.com/intersystems/iris:2020.3.0.221.0
```

The `image:` field specifies the registry, repository, and tag of the InterSystems IRIS image from which to deploy data node containers; the example above specifies an InterSystems IRIS image from the InterSystems Container Registry (ICR).

```
preferredZones:
  - us-east1-b
  - us-east1-c
```

Optionally, use the `preferredZones:` field to specify the zone or zones in which data nodes are to be deployed, as follows:

- If `mirrored: true` and at least two zones are specified, Kubernetes is discouraged from deploying both members of a failover pair in the same zone, which maximizes the chances that at least one is available.

  Note: Bear in mind that deploying the members of a failover pair in separate zones is likely to slightly increase latency in the synchronous communication between them. In addition, specifying multiple zones for the data nodes means that all of the primaries might not be deployed in the same zone, resulting incommunication between the data nodes also being affected by this slightly increased latency. Finally, if compute nodes are included in the cluster, specifying multiple zones for data nodes is very likely to result in some compute nodes being deployed in a separate zone from their associated data nodes, increasing latency in those connections as well.

  Under most circumstances these interzone latency effects will be neglible, but with some demanding workloads involving high message or query volume, performance may be affected. If after researching the issue of interzone connections on your Kubernetes platform and testing your application thoroughly you are concerned about this performance impact, consider specifying a single zone for your mirrored data nodes.

  Regardless of the zones you specify here, you should use the `preferredZones:` field in the `arbiter:` definition (as illustrated later in this section) to deploy the arbiter in a separate zone of its own, which also helps optimize mirror availability.

- The data nodes of an unmirrored cluster are typically deployed in the same zone to minimized latency. If `mirrored: false` and your Kubernetes cluster includes multiple zones, you can use `preferredZones` to follow this practice by specifying a single zone in which to deploy the data nodes.

Kubernetes attempts to deploy in the specified zones, but if this is not possible, deployment proceeds rather than failing.

```
podTemplate:
  spec:
    imagePullSecrets:
      - name: intersystems-pull-scret
```

The `imagePullSecrets` field identifies the Kubernetes secret containing the credentials neededto pull the image specified by the `image:` field; update the `name;` field with the name of the InterSystems IRIS image pull secret you created.

```
      updateStrategy:
        type: RollingUpdate
```

The Kubernetes update strategy type `RollingUpdate` is required for the iriscluster.

```
      storage:
        accessModes:
          - ReadWriteOnce
```

The storage section defines the persistent volume claim required for each data node. The `accessModes:` field specifies the access mode for the persistent volume claim, which must be RWO to enforce a one-to-one relationship between each pod and the persistent volume claim used by the InterSystems IRIS container in the pod for persistent storage.

```
        resources:
          requests:
            storage: 100G
```

The `storage:` field defines the amount of storage (expressed as any unit between kilobytes and exabytes) available to the persistent volume claim. The amount of storage required by data nodes is determined during the cluster planning process and should include a comfortable margin for the future growth of your workload.

```
        storageClassName:
```

The `storageClassName:` field identifies the storage class used for the persistent volume claims; enter the name of the storage class you created as its value.

```
    compute:
      image: containers.intersystems.com/intersystems/iris:2020.3.0.221.0
```

The `compute:` definition shares several fields with the `data:` definition. The values for most are the same, for example the `image:` field specifying the InterSystems IRIS image spec.

```
      replicas: 2
```

The `replica:` field specifies the number of compute nodes to deploy, which must be a multiple of the number of data nodes deployed (the `shards:` field), including `shards:` x 0 and `shards:` x 1. Compute nodes can be added to or removed from the deployed cluster by changing this setting and reapplying the definition, but `replicas:` must always be a multiple of `shards:`.

```
      preferredZones:
        - us-east1-b
        - us-east1-c
```

The `preferredZones:` value for compute nodes should match that for data nodes to ensure that the compute nodes are deployed in the same zone or zones as the data nodes. (Be sure to see the note about interzone latency in the description of the `data:` section, above.)

```
      podTemplate:
        spec:
          imagePullSecrets:
            - name: dockerhub-secret
```

Like the `image:` field's value, the `name:` field of `imagePullSecrets:` is the same as in the `data:` section..

```
      storage:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 10G
        storageClassName: iris-ssd-storageclass
```

The `storage:` settings are also the same as those in the `data:` section, except the amount of storage for compute nodes as specified in the `storage:` field should be kept to a bare minimum to conserve resources. You may even want to create a separate storage class for compute nodes and specify it in the `storageClassName:` field.

```
  arbiter:
      image: containers.intersystems.com/intersystems/arbiter:2020.3.0.221.0
      preferredZones:
        - us-east1-d
      podTemplate:
        spec:
          imagePullSecrets:
            - name: dockerhub-secret
```

The `arbiter:` section contains just three settings:

- The `image:` field specifying the registry, repository, and tag of the image from which to deploy the arbiter. This is different from the image for data and compute nodes, as the arbiter is not an InterSystems IRIS instance.
- The secret containing the credentials required to pull the image, specified by the `name:` field in `imagePullSecrets:`. Typically, the arbiter and Intersystems IRIS images would be in the same repository, so you would specify the same secret here as in the data node and compute node sections. It is possible, however, that the arbiter image could be in a different repository, in which case you would need to create a different image pull secret and specify it for the arbiter.
- The `preferredZones:` setting, which you should use to deploy the arbiter in a separate zone from those specified for the data node primaries and backups, as described in the description of the `data:` section above, in order to optimize mirror availability.

20 Deploy the iriscluster

Once the definition file (for example my-iriscluster-definition.yaml) is complete, deploy the iriscluster with the following command:

```
$ kubectl apply -f my-iriscluster-definition.yaml
iriscluster.intersystems.com/my-iriscluster created
```

Because the IKO extends Kubernetes to add iriscluster as a custom resource, you can apply commands directly to your cluster. For example, if you want to see its status, you can execute the **kubectl get** command on the iriscluster, as in the following:

```
$ kubectl get irisclusters
NAME             DATA    COMPUTE    MIRRORED    STATUS      AGE
my-iriscluster   2       2          true        Creating    28s
```

Follow the progress of cluster creation by displaying the status of the pods that comprise the deployment, as follows:

```
$ kubectl get pods
NAME                                             READY    STATUS     RESTARTS    AGE
intersystems-iris-operator-6499fbbf4-s74lk       1/1      Running    1           1h23m
my-iriscluster-arbiter-0                         1/1      Running    0           36s
my-iriscluster-data-0-0                          0/1      Running    0           28s

...

$ kubectl get pods
NAME                                             READY    STATUS               RESTARTS    AGE
intersystems-iris-operator-6499fbbf4-s74lk       1/1      Running              1           1h23m
my-iriscluster-arbiter-0                         1/1      Running              0           49s
my-iriscluster-data-0-0                          0/1      Running              0           41s
my-iriscluster-data-0-1                          0/1      ContainerCreating    0           6s

...

$ kubectl get pods
NAME                                             READY    STATUS     RESTARTS    AGE
intersystems-iris-operator-6499fbbf4-s74lk       1/1      Running    1           1h35m
my-iriscluster-arbiter-0                         1/1      Running    0           10m
my-iriscluster-compute-0                         1/1      Running    0           10m
my-iriscluster-compute-1                         1/1      Running    0           9m
my-iriscluster-data-0-0                          1/1      Running    0           12m
```

```
my-iriscluster-data-0-1                      1/1     Running   0          12m
my-iriscluster-data-1-0                      1/1     Running   0          11m
my-iriscluster-data-1-1                      1/1     Running   0          10m
```

In the event of an error status for a particular pod, you can examine its log, for example:

```
$ kubectl logs my-iriscluster-data-0-1
```

21 Open the cluster's Management Portal

To load the cluster's Management Portal in your browser, first list the service corresponding to node 1, which can be identified by its name ending in **`data-0-0`**, as shown in the preceding examples. Use a command like the following:

```
$ kubectl get svc my-iriscluster my-iriscluster-data-0-1
NAME            TYPE          CLUSTER-IP    EXTERNAL-IP     PORT(S)                          AGE
my-iriscluster  LoadBalancer  10.35.245.6   35.196.145.234  1972:30011/TCP,52773:31887/TCP   46m
```

Next, load the following URL in your browser, substituting the listed external IP address for the one shown here:

```
http://35.196.145.234:52773/csp/sys/UtilHome.csp
```

The cluster's external IP address is used with the superserver and webserver ports (1972 and 52773, respectively) for all connections to the cluster for data ingestion, queries, and other purposes.

22 Investigate iriscluster deployment errors

The following **kubectl** commands may be particularly helpful in determining the reason for a failure during deployment. Each command is linked to reference documentation at kubernetes.io, which provides numerous examples of these and other commands that may also be helpful.

- **kubectl explain** *resource*

  Lists the fields for the specified resource — for example node, pod, service, persistentvolumeclaim, storageclass, secret, and so on— providing for each a brief explanation and a link to further documentation. This list is useful in understanding the field values displayed by the commands that follow.

- **kubectl describe** *resource* **[***instance-name***]**

  Lists the fields and values for all instances of the specified resource, or for the specified instance of that resource. For example, **kubectl describe pods** shows you the node each pod is hosted by, the containers in the pod and the names of their data volumes (persistent volume claims), and many other details such as the license key and pull secrets.

- **kubectl get** *resource* **[***instance-name***] [***options***]**

  Without options, lists basic information for all instances of the specified resource, or for a specified instance of that resource. However, **kubectl get -o** provides many options for formatting and selecting subsets of the possible output of the command. For example, the command **kubectl get iriscluster -o yaml** *iriscluster-name* output option displays the details fields by the .yaml definition file for the specified iriscluster in the same format with their current values. This allows you, for instance, to create a definition file matching an iriscluster that has been modified since it was created, as these modifications are reflected in the output.

- **kubectl logs (***pod-name* | *resource/instance-name***) [-c** *container-name***]**

  Displays the logs for the specified container in a pod or other specified resource instance (for example, **kubectl logs deployment/***intersystems-operator-name*). If a pod includes only a single container, the **-c** flag is optional. (For more log information, you can use **kubectl exec** to examine the messages log of the InterSystems IRIS instance on a data or compute node, as described in the next entry.)

- **kubectl exec** (*pod-name* | *resource*/*instance-name*) **[-c** *container-name*] **--** *command*

  Executes a command in the specified container in a pod or other specified resource instance. If *container-name* is not specified, the command is executed in the first container, which in an iriscluster pod is always the InterSystems IRIS container of a data or compute node. For example, you could use **kubectl exec** in these ways:

  – **kubectl exec** *pod-name* **-- iris list**

    Displays information about the InterSystems IRIS instance running in the container.

  – **kubectl exec** *pod-name* **-- more /irissys/data/IRIS/mgr/messages.log**

    Displays the instance's messages log.

  – **kubectl exec** *pod-name* **-it -- iris terminal IRIS**

    Opens the InterSystems Terminal for the instance.

  – **kubectl exec** *pod-name* **-it -- "/bin/bash"**

    Opens a command line inside the container.

23 Remove the iriscluster

To fully remove the cluster, you must use **kubectl** to delete not only the cluster, but also the persistent volume claims and (typically) the service associated with it. For example:

```
$ kubectl delete -f my-iriscluster-definition.yaml
iriscluster.intersystems.com "my-iriscluster" deleted
$ kubectl delete pvc —all
persistentvolumeclaim "iris-data-my-iriscluster-compute-0" deleted
persistentvolumeclaim "iris-data-my-iriscluster-compute-1" deleted
persistentvolumeclaim "iris-data-my-iriscluster-data-0-0" deleted
persistentvolumeclaim "iris-data-my-iriscluster-data-0-1" deleted
persistentvolumeclaim "iris-data-my-iriscluster-data-1-0" deleted
persistentvolumeclaim "iris-data-my-iriscluster-data-1-1" deleted
$ kubectl delete svc iris-svc
service "iris-svc" deleted
```

You can also fully remove the iriscluster, as well as the IKO, by unprovisioning the Kubernetes cluster on which they are deployed. The operator can be deleted without unprovisioning the cluster by issuing the command **helm uninstall intersystems**.