
# Using the InterSystems Kubernetes Operator (Version 3.5)

2024-05-06

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

*Using the InterSystems Kubernetes Operator (Version 3.5)*
InterSystems IRIS Data Platform 2024-05-06
Copyright © 2024 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble® InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

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

# Using the InterSystems Kubernetes Operator (Version 3.5)

This page explains how to use the InterSystems Kubernetes Operator (IKO) to deploy sharded clusters and other InterSystems IRIS configurations 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?

In Kubernetes, a resource is an endpoint that stores a collection of API objects of a certain kind, from which an instance of the resource can be created or deployed as an object on the cluster. For example, built-in resources include, among many others, *pod* (a set of running containers), *service* (a network service representing an application running on a set of pods), and *persistent volume* (a directory containing persistent data, accessible to the containers in a pod).

The InterSystems Kubernetes Operator (IKO) extends the resources built into the Kubernetes API with a custom resource called *IrisCluster*, representing an InterSystems IRIS cluster. An instance of this resource — that is, a sharded cluster, or a standalone InterSystems IRIS instance, optionally configured with application servers in a distributed cache cluster — can be deployed on any Kubernetes platform on which the IKO is installed and benefit from all the features of Kubernetes such as its services, role-based access control (RBAC), and so on.

The IrisCluster resource 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, examine your use case and answer these questions:

- Is containerized deployment of InterSystems IRIS and your application the best approach?
- Have you identified one or more suitable Kubernetes platforms on which to deploy your containerized InterSystems-IRIS based application? For example, major public cloud 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.
- Which IrisCluster topology best suits your use case?
  - A sharded cluster

    The InterSystems IRIS sharding architecture can help you scale for data volume and optimize performance for demanding workloads. A sharded cluster can consist of data nodes only, or also include compute nodes for workload separation and increased query throughput. The data nodes can be mirrored for high availability or nonmirrored.
  - A distributed cache cluster

    InterSystems IRIS distributed caching can help you scale for user volume by distributing application connections across multiple application servers. The cluster's data server can be mirrored or nonmirrored.
  - A standalone instance

    Many applications run on a single instance of InterSystems IRIS, which is often mirrored for high availability.

For detailed information about defining your IrisCluster's topology, see Define the IrisCluster topology.

# 4 Plan your deployment

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

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

For detailed information about InterSystems IRIS sharded clusters, see Horizontally Scaling for Data Volume with Sharding in the *Scalability Guide*. If you are instead deploying a distributed cache cluster, there is important information to review in Horizontally Scaling for User Volume with Distributed Caching in the same 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.

**Note:** The IKO is compatible with InterSystems IRIS 2021.2 and later versions, and with Kubernetes 1.24 and later versions. Use with the most recent compatible versions is recommended.

# 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 or distributed cache 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
gcloudPreparing 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-3.5.0.47.0-unix.tar.gz, from the InterSystems Worldwide Response Center (WRC) download area and extract its contents. Next, upload the extracted directory, with the same base name as the archive file (for example iris_operator-3.5.0.47.0) 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.
- A README file, which recaps the steps needed to obtain and install the IKO.

# 10 Locate the IKO image

To install the IKO, Kubernetes must be able to download (**docker 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:3.5.0.47.0`**, 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-3.5.0.47.0/image/iris_operator-3.5.0.47.0-docker.tgz
fd6fa224ea91: Loading layer [==================================================>] 3.031MB/3.031MB

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

Loaded image: intersystems/iris-operator:3.5.0.47.0
$ docker images
REPOSITORY                 TAG         IMAGE ID      CREATED       SIZE
intersystems/iris-operator  3.5.0.47.0  9a3756aed423  3 months ago  77.3MB
$ docker tag intersystems/iris-operator:3.5.0.47.0 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 to 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 chart files

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
```

You can also adjust the behavior of all InterSystems IRIS clusters and standalone instances deployed by the IKO by changing the values of the operator environment variables declared in the values.yaml file. The `operator:` section indicates the default values of the variables supported for use with the IKO, as shown:

```
operator:
  registry: containers.intersystems.com
  repository: intersystems/iris-operator-amd  tag: 2.0.0

  # Operator Environment Variables
  useFQDN: true
  webserverPort: "52773"
```

- `webserverPort` — The default web server port, used to connect to an InterSystems IRIS instance's Management Portal and for other purposes, is 52773. To change this for all instances deployed by the IKO, set the`webserverPort` variable to the desired port (for example, 53773).
- `useFQDN` — By default, the IKO uses FQDN hostnames, for example **`my-IrisCluster-data-0-0.iris-svc.us-west1.svc.cluster.local`**, because including the governing service allows the hosts to find each other. However, non-FQDN hostnames such as **`my-IrisCluster-data-0-0`** are easier to read and may be required in certain contexts (for example, when using custom DNS). To use non-FQDN hostnames, set the `useFQDN` variable to `false`. When you do this, the IKO creates a DNSConfig which adds the cluster domain to the DNS search space, allowing pods to communicate with one another using non-FQDN hostnames, and configures non-FQDN hostnames for all InterSystems IRIS node types (`data`, `compute`, `arbiter`, `webgateway`).

# 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-3.5.0.47.0/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"
```

Add the **--watch** option to the command so you can wait until the status of the operator changes to ready:

```
$ kubectl get deployments -l "release=intersystems, app=iris-operator" --watch
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
intersystems-iris-operator   0/1     1            0           30s
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
intersystems-iris-operator   1/1     1            1           60s
```

**Note:** Because the IKO is designed to support as wide a variety of Kubernetes environments as possible, you may see warnings regarding deprecated elements, such as the following, during the installation process:

```
W0616 10:23:55.628201 622222 warnings.go:67] apiregistration.k8s.io/v1beta1
  APIService is deprecated in v1.19+, unavailable in v1.22+;
  use apiregistration.k8s.io/v1 APIService
```

These warnings can be safely ignored.

# 14 Define the IrisCluster topology

An IrisCluster can be deployed as a sharded cluster, a distributed cache cluster, or a standalone InterSystems IRIS instance. All three topologies can be mirrored. Compute nodes can optionally be added to a sharded cluster, and application servers are added to a standalone instance to create a distributed cache cluster. As described in detail in the following section, the topology deployed, including additional node types that can be added, is determined by the node definitions in the `topology` section of the definition file, as follows:

- If the data node definition (`topology`/`data`) contains the `shards` field, a sharded cluster is deployed, with the number of data nodes specified by `shards`. Otherise, a standalone instance is deployed.
- If the `mirrored` field is included in `data` with the value `True`, the data nodes or standalone instance are mirrored; otherwise, they are nonmirrored. When deploying mirrored nodes, you can add an arbiter node using the `topology`/`arbiter` definition.
- Compute nodes can be added to a sharded cluster using `topology`/`compute`. If a standalone instance is deployed, use `compute` to add application servers, making it the data server of a distributed cache cluster.
- Web server nodes, each of which hosts a web server and the InterSystems Web Gateway, can be added to your IrisCluster using the `webgateway` definition.

  **Important:** Best practices for both sharded and distributed cache clusters include distributing application query connections across multiple InterSystems IRIS nodes. While an IrisCluster does not include a Kubernetes-based mechanism for this on deployment, deployed web server nodes do provide such a mechanism, since the InterSystems Web Gateway can distribute connections across multiple InterSystems IRIS instances defined in its server access configuration (for more information, see Prepare the Web Gateway configuration files). For this reason, including one or more web server nodes is the simplest way to distribute application connections in the recommended manner. You can, however, deploy a custom load balancer on the Kubernetes cluster hosting the IrisCluster (which is easier on some Kubernetes platforms than others) or use a third party-tool to distribute connections.

- You can also add the following types of nodes, using the appropriate definition within the `topology` section:
  - InterSystems System Alerting and Monitoring (SAM), using the `sam` definition.
  - InterSystems API Manager (IAM), using the `iam` definition.

# 15 Understand IrisCluster node naming

The naming of IrisCluster nodes is determinate, allowing you to plan connections before deployment (for example, identifying the nodes to be included in the server access configuration of each `webgateway` node). Names assigned to IrisCluster nodes always follow the same pattern, which is *clustername*`-`*nodetype*`-`*N*. For example, in an unmirrored IrisCluster called **`myCluster`**, the following nodes are called **`myCluster-data-0`**, depending on the configuration:

- Data node 1 of a sharded cluster
- The data server of a distributed cache cluster
- A standalone instance

If mirrored, the primary (as deployed) would be **`myCluster-data-0-0`** and the backup **`myCluster-data-0-1`**.

Other nodes are named in the same format (but without the mirroring variation), for example the first compute node is *clustername*`-compute-0`. Therefore, if an IrisCluster named **`MyCluster`** was a sharded cluster with four mirrored data nodes, four compute nodes, an arbiter, and three webgateway nodes, they would be named as shown in the following table.

| | | | |
|---|---|---|---|
| **`myCluster-data-0-0`** | **`myCluster-compute-0`** | **`myCluster-arbiter-0`** | **`myCluster-webgateway-0`** |
| **`myCluster-data-0-1`** | **`myCluster-compute-1`** | | **`myCluster-webgateway-1`** **`myCluster-webgateway-2`** |
| **`myCluster-data-1-0`** | **`myCluster-compute-2`** | | |
| **`myCluster-data-1-1`** | **`myCluster-compute-3`** | | |
| **`myCluster-data-2-0`** | | | |
| **`myCluster-data-2-1`** | | | |
| **`myCluster-data-3-0`** | | | |
| **`myCluster-data-3-1`** | | | |

**Note:** If you included one or more DR async members in each mirror using the `mirrorMap` field, they would be named **`myCluster-data-0-3`** and so on, in sequence.

# 16 Plan persistent volumes

Once you know which nodes will be included in your deployment, you can plan the required storage volumes.

Kubernetes provides the durable storage needed by containerized programs in the form of persistent volumes, which exist independently of the individual pods that use them so that the data they store, for example application or custom configuration data, remains accessible regardless of the status of any containers or pods. A persistent volume claim is a specification for a persistent volume, including such characteristics as access mode, size, and storage class, that can be selected when requesting one or more volumes.

The IKO provides predefined persistent volumes for `data`, `compute`, `webgateway`, `sam`, and `iam` nodes. Typically these are sufficient for an IrisCluster deployment, but you may well want to increase the size of some of them from the default of 2 GB based on your cluster planning process. You can override the size and/or other default characteristics of one or more of the predefined volumes by modifying the appropriate specification in the node type definition — for example, the `storageDB` specification for data, compute, and webgateway nodes.

You can also create custom persistent volume claims using the `volumeClaimTemplates` field, and specify them when adding custom persistent volumes to data and compute nodes using the `volumeMounts` field. (You can also use this field to mount ephemeral storage volumes, as defined in the `volumes` field.)

# 17 Create the IrisCluster definition file

You are now ready to create your IrisCluster definition YAML file. The following sections provide:

- A listing of all fields defined in the IrisCluster custom resource definition (CRD) that can be used in a definition file. Each field is linked to the information you need to use it. Required fields are indicated as such.
- The contents of the iris-sample.yaml file in the /Samples directory, which is a customizable sample IrisCluster definition file containing commonly used fields. To assist you in selecting the ones you want to include, most of the contents are commented out, with a brief explanation and with each field linked to the information about it, as in the CRD,

- An explanation of each field, suggestions for customizing it, and instructions for any actions you need to take before using it, including creating needed Kubernetes objects. For example, the section about the `licenseKeySecret` field explains that you must create a Kubernetes secret containing the InterSystems IRIS license key and then specify that secret by name in the field.

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

## 17.1 Review the IrisCluster custom resource definition (CRD)

```
apiVersion: intersystems.com/v1alpha1
kind: IrisCluster
metadata:
  name: name
spec:
  licenseKeySecret:
    name: name
  configSource:
    name: name
  imagePullSecrets:
  - name: name
  - ...
  storageClassName: name
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  volumeClaimTemplates:
  - metadata:
      name: nameN
    spec:
      accessModes:
      - {ReadWriteOnce|ReadOnlyMany|ReadWriteMany}
      resources:
        requests:
          storage: size
      storageClassName: nameN
  - ...
  volumes:
  - name: nameN
    type:
      typeName: name
  - ...
  serviceTemplate:
    spec:
      type: LoadBalancer
      externalTrafficPolicy: Local
      ports:
      - name: name
        port: portnumber
      - ...
  tls:
    connection_type:
      volume_source:
        volume_source_spec: common-certs
        ...
    ...
  topology:
    data:
      image: registry/repository/image:tag
    [common InterSystems IRIS node fields, optional for all node types]
      updateStrategy:
        type: {RollingUpdate|OnDelete}
      preferredZones:
        - zoneN
        - ...
      podTemplate:
        core.PodTemplateSpec
      compatibilityVersion: iris-version
    [end common InterSystems IRIS node fields]
      shards: N
      mirrored: {true|false}
      mirrorMap: primary,backup,drasync, ... drasync
      mirrorName: basename
      storage{DB|WIJ|Journal1|Journal2}:
        resources:
          requests:
            storage: size
        storageClassName:
        mountPath: path
      volumeMounts:
      - mountPath: pathN
        name: volumeClaimTemplateN or volumeN
      - ...
      containers:
      - name: containerName
        image: registry/repository/image:tag
        field: value
        ...
      - ...
    compute:
      image: registry/repository/image:tag
     see common InterSystems IRIS node fields in data definition
      replicas: N
      storage{DB|WIJ|Journal1|Journal2}:
        resources:
          requests:
```

```
          storage: size
      storageClassName:
    volumeMounts:
    - mountPath: pathN
      name: volumeClaimTemplateN or volumeN
    - ...
    containers:
    - name: containerName
      image: registry/repository/image:tag
      field: value
      ...
    - ...
  arbiter:
    image: registry/repository/image:tag
    see common InterSystems IRIS node fields in data definition
  webgateway:
    image:  registry/repository/image:tag
    see common InterSystems IRIS node fields in data definition
    type: {apache|apache-lockeddown|nginx}
    replicas: N
    applicationPaths:
      - pathN
      - ...
    alternativeServers: {FailOver|LoadBalancing}
    storageDB:
      resources:
        requests:
          storage: spec
      storageClassName:
    volumeMounts:
    - mountPath: pathN
      name: volumeClaimTemplateN or volumeN
    - ...
  sam:
    image: registry/repository/image:tag
    see common InterSystems IRIS node fields in data definition
    replicas: {0|1}
    storageSam:
      resources:
        requests:
          storage: spec
    image[AlertManager|Grafana|Nginx|Prometheus]: registry/repository/image:tag
    loginSecret:
      name: secret-name
  iam:
    image: registry/repository/image:tag
    see common InterSystems IRIS node fields in data definition
    replicas: {0|1}
    storagePostgres:
      resources:
        requests:
          storage: spec
    imagePostgres: registry/repository/image:tag
```

## 17.2 Review the sample IrisCluster definition file

```
## uncommented fields deploy one InterSystems IRIS data server

## WARNING: default password is not reset, to do so include
## configSource below

## include commented fields for purposes described; see documentation at
## https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=AIKO3535_clusterdef_sample

## update image tags (from ":tag") before using; see list of available images at
## https://docs.intersystems.com/components/csp/docbook/Doc.View.cls?KEY=PAGE_containerregistry

   apiVersion: intersystems.com/v1alpha1
   kind: IrisCluster
   metadata:
     name: sample
   spec:

## provide InterSystems IRIS license key if required
#     licenseKeySecret:
#       name: iris-key-secret

## specify files used to customize the configurations of
## InterSystems IRIS nodes, including passwordHash parameter
## to set the default password, securing InterSystems IRIS
#     configSource:
#       name: iris-cpf

## provide repository credentials if required to pull images
#     imagePullSecrets:
#       - name: iris-pull-secret

## provide VolumeSource specifications for certificates for each desired TLS feature
## "common" enables all TLS features, but each particular feature's property is given priority over
"common"
#     tls:
#       common:
#         secret:
#           secretName: tls-certs
#       mirror:
#         csi:
#           driver: secrets-store.csi.k8s.io
#           readOnly: true
#           volumeAttributes:
#             secretProviderClass: "my-provider"
#       webgateway:
#         secret:
#           secretName: wg-certs
#       superserver:
#         secret:
#           secretName: superserver-certs
#       ecp:
#         secret:
#           secretName: ecp-certs
#       iam:
#         secret:
#           secretName: iam-certs

## specify platform-specific storage class used to allocate storage
## volumes (default: use platform-defined class)
#     storageClassName: iris-ssd-storageclass

## select update strategy (default: RollingUpdate)
#     updateStrategy:
#       type: RollingUpdate

## create external IP address(es)for the cluster
## ("type: LoadBalancer" and "externalTrafficPolicy: Local" are required)
#     serviceTemplate:
#       spec:
#         type: LoadBalancer
#         externalTrafficPolicy: Local

## define persistent volumes (to be mounted by "VolumeMounts:" in node definitions)
#   volumeClaimTemplates:
#     - metadata:
#         name: extra-disk
#       spec:
#         accessModes:
#           - ReadWriteOnce
#         resources:
#           requests:
#             storage: 2Gi

## define ephemeral volumes (to be mounted by "VolumeMounts:" in node definitions)
#   volumes:
```

```
#    - name: my-secret
#      secret:
#        secretName: my-secret
#    - name: my-secret2
#      secret:
#        secretName: my-secret2
#    - name: my-config
#      configMap:
#        name: my-config

## topology: defines node types to be deployed; only data: is required

     topology:
       data:
         image: containers.intersystems.com/intersystems/iris:tag

## deploy a sharded cluster of data nodes and (optionally) compute
## nodes; if not included, "data:" definition in "topology:" deploys
## a single data server, "compute:" adds application servers
#          shards: 2

## deploy mirrored data nodes or data server (default: nonmirrored)
#          mirrored: true

## override default size and other attributes of predefined storage
## volumes for data nodes (additional volume names: storageWIJ,
## storageJournal1, storageJournal2); can also be included in
## "compute:" definition
#          storageDB:
#            resources:
#              requests:
#                storage: 10Gi
#            storageClassName: my-storageclass

## constrain nodes to platform-specific availability zones (can be
## included in other node definitions)
#          preferredZones:
#            - us-east1-a
#            - us-east1-b

## mount volumes defined in "volumeClaimTemplates:" (persistent) and "volumes:" (ephemeral)
#          volumeMounts:
#            - mountPath: "/extra-disk"
#              name: extra-disk
#            - mountPath: "/my-secret"
#              name: my-secret
#            - mountPath: "/my-config"
#              name: my-config

## deploy compute nodes, or application servers if "shards:" not included;
## use "replicas:" to specify how many
#        compute:
#          image: containers.intersystems.com/intersystems/iris:tag
#          replicas: 2

## mount ephemeral volume defined in "volumes:"
#          volumeMounts:
#            - mountPath: "/my-secret2"
#              name: my-secret2

## specify version of InterSystems IRIS containers if not default of 2022.2.0
#          compatibilityVersion: "2022.3.0"

## deploy arbiter for mirrored data nodes (or data server)
#        arbiter:
#          image: containers.intersystems.com/intersystems/arbiter:tag

## deploy webgateway (web server) nodes
#        webgateway:
#          image: containers.intersystems.com/intersystems/webgateway:tag
#          type: apache
#          replicas: 2
#          applicationPaths:
#            - /external
#            - /internal
#          alternativeServers: LoadBalancing

## deploy System Alerting and Monitoring (SAM) with InterSystems IRIS
#       sam:
#         image: containers.intersystems.com/intersystems/sam:tag

## deploy InterSystems API Manager (IAM) with InterSystems IRIS
#       iam:
```

```
#       image: containers.intersystems.com/intersystems/iam:tag
```

## 17.3 apiVersion: Define the IrisCluster

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

**Required**

The first four fields, which define the object you are defining, are required by Kubernetes.

Change the value of the `name` field in `metadata` to the name you want to give the IrisCluster.

The `spec` section contains the nested fields, required and optional, that make up the specification for an IrisCluster deployment.

## 17.4 licenseKeySecret: Provide a secret containing the InterSystems IRIS license key

```
  licenseKeySecret:
    name: name
```

**Optional**

The `licenseKeySecret` field specifies a Kubernetes secret containing the InterSystems IRIS license key to be activated in all of the InterSystems IRIS containers in the cluster.

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
```

**Note:** If you are not deploying a sharded cluster but rather a configuration with a single `data` node (see Define the IrisCluster topology), the license you use does not have to be sharding-enabled.

Finally, update the `name` field in `licenseKeySecret` with the name of the secret you created.

The `licenseKeySecret` field is optional. For example if you are deploying from an InterSystems IRIS Community Edition image, you don't need a license and can omit `licenseKeySecret`.

If you are including a sam node in your deployment, you can add an InterSystems System Alerting and Monitoring (SAM) license to the secret you identify in the `licenseKeySecret` field. To do this, simply add another **--from-file** flag specifying the SAM license (which must be called sam.key) to the **kubectl create secret** command, for example:

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

If you do not include an InterSystems IRIS license key — that is, you include only a SAM key — in the secret you specify in the `licenseKeySecret` field, the data and compute nodes in the deployment will not start.

**Note:** If you want to provide different licenses for the `data` and `compute` nodes, for example because you are deploying an **`irishealth`** image on the data nodes and an **`iris`** image on the compute nodes, rather than using the `licenseKeySecret` field, you can follow these steps:

1. Create two secrets of type **`generic`**, each containing one of the license keys, for example:

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

2. Create two ephemeral volumes using the `volumes` field, each holding one of the secrets, for example::

```
volumes:
  - name: data-secret
    secret:
      secretName: data-secret
  - name: compute-secret
    secret:
      secretName: compute-secret
```

3. In the `topology` definitions for both `data` and `compute` nodes, use the `args` field in the `podTemplate` spec to specify the location of the key, and mount the appropriate ephemeral volume at that location using the `volumeMounts` field, for example:

```
  topology:
    data:
      ...
      podTemplate:
        spec:
          args:
          - --key
          - /datasecret/key/iris.key
      ...
      volumeMounts:
      - mountPath: "/computesecret/key/"
        name: compute-secret
      - ...
    compute:
      ...
      podTemplate:
        spec:
          args:
          - --key
          - /computesecret/key/iris.key
      ...
      volumeMounts:
      - mountPath: "/computesecret/key/"
        name: compute-secret
      - ...
```

This approach can be used more generally to supply varying read-only input data, such as secrets and configmaps, to the different node types.

## 17.5 configSource: Create configuration files and generate a configmap from them

```
configSource:
  name: name
```

**Optional**

The `configSource` field specifies a Kubernetes configmap generated from one or more of the following:

- A configuration merge file called common.cpf used to customize the configurations of InterSystems IRIS cluster nodes (`data` and `compute`) when deployed.
- A configuration merge file called data.cpf used to further customize `data` nodes only; settings in this file override the same settings in common.cpf.

- A configuration merge file called compute.cpf used to further customize `compute` nodes only; settings in this file override the same settings in common.cpf.
- An InterSystems Web Gateway configuration file CSP.ini to be installed on `webgateway` (web server) nodes when deployed, or a CSP-merge.ini merge file to modify the CSP.ini file generated by the IKO. A CSP.conf web server-specific configuration file can also be included.

**Important:** All configuration merge (*.cpf and CSP.*) files are optional, and they can be included in any combination. For effective security, however, you *must* include at least a common.cpf file (or instead both a data.cpf and a compute.cpf file) containing the passwordHash parameter to reset the default InterSystems IRIS password.

Kubernetes configmaps keep your containerized applications portable by separating configuration artifacts, such as the content of these files, from container image content. For an IrisCluster, you generate a configmap from the contents of the configuration files you specify and identify it using the `configSource` field. The IKO then uses the specified configmap during deployment to generate a separate configmap for each node type in the cluster, which in turn is used to create a local configuration file on each node of that type.

**Note:** Some of the configuration performed by the IKO, for example mirror configuration, uses settings that can be specified in merge files; any settings specified by the IKO are overwritten by the same parameters in user-provided .cpf files.

To use configuration merge files to customize the configurations of the IrisCluster's InterSystems IRIS nodes (`data` and `compute`), provide your own Web Gateway configuration file for the `webgateway` nodes, or both, you should:

- Prepare the configuration merge files
- Set the default InterSystems IRIS password
- Prepare the Web Gateway configuration files
- Create the Kubernetes configmap
- Update `configSource`
- Modify configmaps to reconfigure nodes, as needed

## 17.5.1 Prepare the configuration merge files

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 configuration merge feature allows you to specify a merge file that overwrites the values of one or more settings in, or adds settings to, the default CPF that comes with an instance when it is deployed. For details, see Automating Configuration of InterSystems IRIS with Configuration Merge.

To use configuration merge when deploying your IrisCluster, customize the template common.cpf, data.cpf, and compute.cpf files provided in the samples/ directory by adding the CPF settings you want to apply to all InterSystems IRIS nodes, the data nodes, and the compute nodes (if included) respectively. The provided common.cpf template file contains only a sample passwordHash setting, described in the next section; the data.cpf and compute.cpf template files contain only a sample SystemMode setting, which displays text on the InterSystems IRIS Management Portal.

For numerous helpful examples of parameters you might customize for several purposes, see Useful Parameters for Automated Deployment in *Automating Configuration of InterSystems IRIS with Configuration Merge*. For example, the data nodes in a sharded cluster must be configured to allocate a database cache of the appropriate size, as calculated by you before

deployment; the following illustrates how you would add the **[config]** section `globals` parameter to the data.cpf file to configure the database caches of the data nodes to be 20 GB (the globals setting is in MB):

```
[StartUp]
SystemMode=my-IrisCluster
[config]
globals= 0,0,20480,0,0,0
```

### 17.5.2 Set the default InterSystems IRIS password

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 all InterSystems IRIS containers. For this reason, even if you have no other reason to provide configuration merge files (which are optional), you should include at least a common.cpf containing the passwordHash parameter, as illustrated in the provided template common.cpf, to reset the default password on all InterSystems IRIS nodes, for example:

```
[Startup]
PasswordHash=dd0874dc346d23679ed1b49dd9f48baae82b9062,10000,SHA512
```

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 about this, the passwordHash parameter, and the default password, see Authentication and Passwords in *Running InterSystems Products in Containers*.

**Important:** The passwordHash parameter does not change the default password for the CSPSystem account, which is used by InterSystems IRIS and the local Web Gateway instance installed with it to communicate with each other, and by default provides management access to the Web gateway. For more information, see the following section.

### 17.5.3 Prepare the Web Gateway configuration files

As described in Configure the Web Gateway in the *Web Gateway Configuration Guide*, the Web Gateway's configuration is managed using the Web Gateway management pages, but contained in the CSP.ini file (much as an InterSystems IRIS instance's configuration is contained in the iris.cpf file). The CSP.ini file you specify in your configmap is automatically installed on every `webgateway` node deployed. You have two options for specifying this file:

- Supply your own CSP.ini. If you have experience with the Web Gateway, you can use a CSP.ini from an existing installation as a template and prepare one to be installed on your webgateway nodes by the IKO.
- Let the IKO generate a CSP.ini. When you do this, you can include in your configmap a merge file called CSP-merge.ini to modify the IKO-generated CSP.ini. This file works in the same way as the .cpf files described in Prepare the configuration merge files, that is, by replacing values of existing parameters in, or adding parameter/value pairs to, the IKO-generated CSP.ini.

  **Important:** While the CSP-merge.ini file is not required with an IKO-generated CSP.ini, InterSystems recommends including it in all clusters with multiple Web Gateway nodes because it provides the only means of updating the configurations of all of them in a single operation, as described in Updating the configuration of multiple Web Gateways.

**Note:** You can also supply your own CSP.conf file containing the Web Gateway's Apache or Nginx-specific configuration, but this is note typically done when an IKO-generated CSP.ini file is used, in which case the IKO also makes the needed changes in the default CSP.conf file.

If your deployment includes multiple `webgateway` nodes, bear in mind the information in the following sections when deciding whether to supply your own configuration files or allow the IKO to generate them:

- Populating the server access configuration in the CSP.ini file

- Updating the configuration of multiple Web Gateways
- Changing the Web Gateway's default password
- Changing the credentials for remote servers

### Populating the server access configuration in the CSP.ini file

Deployed web server (`webgateway`) nodes provide a means of distributing application connections within the IrisCluster specification. (You can also deploy a custom load balancer on the Kubernetes cluster hosting the IrisCluster or use a third party-tool to distribute connections.) For this reason, ensure that the server access configuration in the configuration file, which specifies the InterSystems IRIS instances to which the Web Gateway directs incoming connections from the web server, is populated according to the relevant best practice for distributing application connections, as follows:

- If the deployment is a sharded cluster, add all data nodes and compute nodes (if any) to the pool.
- If the deployment is a distributed cache cluster, add all compute (application server) nodes to the pool.
- If the deployment is a standalone instance, it should be the only remote server in the pool.

**Important:** Web Gateway connections to `mirrored` data nodes are automatically mirror aware, that is, connections are always to twhichever failover member is the current primary.

As noted in Understand IrisCluster node naming, it is possible to populate the server access configuration before deployment because the names assigned to IrisCluster nodes are determinate, following the pattern *clustername*`-`*nodetype*`-`*N* (or *clustername*`-data-`*N*`-`*N* in the case of mirrored data nodes). For example, in a sharded cluster with four mirrored data nodes and four compute nodes, the server access configuration would include the four data node primaries and four compute nodes, which would be named as follows:

| | |
|---|---|
| **myCluster-data-0-0** | **myCluster-compute-0** |
| **myCluster-data-1-0** | **myCluster-compute-1** |
| **myCluster-data-2-0** | **myCluster-compute-2** |
| **myCluster-data-3-0** | **myCluster-compute-3** |

If you do not include a CSP.ini file in your configmap, the IKO will generate one appropriate for the cluster and install it on every webgateway node, populating the server access configuration as described in the foregoing. The IKO can also generate a CSP.conf file if you do not provide one.

**Important:** InterSystems recommends the use of TLS to secures all connections in your deployment. The CSP.ini file generated by the IKO secures connections between webgateway nodes and data and compute nodes if you so specify in the tls section of the IrisCluster definition. If you provide your own CSP.ini and prepopulate the server access configuration, you can include TLS for each entry; if you do not prepopulate, you can configure it when you populate the server access configurations using the Web Gateway management pages on the individual nodes.

### Updating the configuration of multiple Web Gateways

As described in Connect to the IrisCluster, the `serviceTemplate` field creates one or more Kubernetes services that expose the first stateful set in the pod for each deployed node type to the network through an external IP address. This includes webgateway nodes, which means that if you have deployed just one, you can always access its management pages to change its configuration by loading the URL `http://`*`external-ip`*`:80/csp/bin/Systems/Module.cxw` in your browser. (For nodes deployed from the **`webgateway-lockeddown`** image, load `http://`*`external-ip`*`:52773/csp/bin/Systems/Module.cxw`.) If you have deployed multiple webgateway nodes and want to make configuration changes across all of them, however, you have these options:

- Modify the Web Gateway settings in the deployed webgateway node configmap using the procedure described in Modify configmaps to reconfigure nodes to propagate the changed to the deployed nodes. The settings already in the configmap depend on whether you supplied your own CSP.ini file, as described in Prepare the Web Gateway configuration files, or had the IKO generate the CSP.ini file and supplied a CSP-merge.ini file to modify it (as recommended), but in either case you can make whatever changes you want by either modifying settings already in the configmap or adding new ones.
- Create a similar service for each of the webgateway nodes that exposes it through an external IP address, then use the URL above to make the changes on each node separately.
- Directly modify the CSP.ini file on each pod by using the **kubectl exec** command to open a shell or edit the file within the container.

**Important:** Be sure to review information about securing Web Gateway connections in Changing the Web Gateway's default password and tls: Configure TLS security.

### Changing the Web Gateway's default password

The predefined **CSPSystem** account on an InterSystems IRIS instance is used for communication between an InterSystems IRIS instance and the local InterSystems Web Gateway instance installed with it, and by default is used for access to the local Web Gateway's configuration through its management pages. For effective security, you must change the default password for this account (along with the other predefined accounts) on both an InterSystems IRIS instance and its local Web Gateway instance either as part of deployment or immediately after; Authentication and Passwords in *Running InterSystems Products in Containers* provides details on this topic and instructions for making these changes in an InterSystems IRIS container.

In the case of a webgateway node deployed by the IKO, if you provide your own CSP.ini file, ensure that if **CSPSystem** (or any other predefined account) is specified as the management access account, the password is not **SYS**; if it is, change it. You can also change the access credentials to any username and password you choose. Additionally, while the Web Gateway encrypts the management access password in the CSP.ini file on first starting up, including it in plain text could expose it on disk or the network, so the best practice is to make sure it is encrypted in the file you provide; you can do this after you change it, if necessary, by starting up a local test Web Gateway instance with your configuration file in place, which encrypts the password.

### Changing the credentials for remote servers

In a separate measure from the Web Gateway management access credentials discussed in Changing the Web Gateway's default password, each entry in a Web Gateway's server access configuration specifies the username and password of an account on the specified InterSystems IRIS instance to be used to authenticate to the instance. If you provide your own CSP.ini, ensure that the specified account is a secure one that does not use the default password (see Authentication and Passwords). You can then specify this account for each remote server in your CSP.ini (clearly it is simplest to use the same account for all).

The CSP.ini generated by the IKO does not configure credentials for the server access configuration entries. To change this, which you must for effective security, use either the second method described above for Updating the configuration of multiple Web Gateways (creating services) or the third (using **kubectl exec**).

## 17.5.4 Create a configmap for the configuration files

Create a Kubernetes configmap specifying the files you have prepared, using a command like this:

```
$ kubectl create cm my-iriscluster-config --from-file common.cpf --from-file data.cpf
    --from-file compute.cpf --from-file CSP.ini --from-file CSP.conf
```

To have the IKO generate the CSP.ini file and include a configuration merge file for the webgateway nodes (as described in Prepare the Web Gateway configuration files) you would replace `--from-file CSP.ini` in the above example with `--from-file CSP-merge.ini`, as follows:

```
$ kubectl create cm my-iriscluster-config --from-file common.cpf --from-file data.cpf
    --from-file compute.cpf --from-file CSP-merge.ini --from-file CSP.conf
```

### 17.5.5 Update configSource:

In the IrisCluster definition file, specify the name of the configmap as the value for `configSource` for the `name` field within `configSource`, for example:

```
  configSource:
    name: my-iriscluster-config
```

If you did not create a configmap, do not specify a value for this field.

### 17.5.6 Modify configmap to reconfigure nodes

To modify the *.cpf or CSP.* settings of nodes in the deployed cluster, edit the node-specific configmap generated by the IKO to make one or more changes to the settings it contains. For instance, if you deploy a cluster called **`my-iriscluster`** and the configmap you specify in its `configsource` field was generated (using the **kubectl create** command as illustrated above) from a data.cpf file, a compute.cpf file, and either a CSP.ini file or a CSP-merge.ini file, you could modify or remove any or all of the settings in the source files, and add new ones, by using a command like the following:

```
kubectl edit configmap my-iriscluster-config
```

To push your changes to the affected *.cpf and/or CSP.* files in the pods, use a **kubectl replace** ommand like the following to force the IKO to reprocess the spec, incorporating the modified configmap.

```
kubectl replace -f my-iriscluster.yaml
```

The **kubectl replace** command reprocesses the definition even when it has not been changed, which is needed in this case because it is the configmap that has been modified, not the definition.

**Note:** In addition to the settings you specified, the contents of the deployed configmaps contain settings added by the IKO. Any settings specified by the IKO are overwritten by the same parameters in user-provided .cpf files.

As described in Updating the configuration of multiple Web Gateways, the settings already in the webgateway node configmap depend on whether you supplied your own CSP.ini file, as described in Prepare the Web Gateway configuration files, or had the IKO generate the CSP.ini file and supplied a CSP-merge.ini file to modify it (as recommended), but in either case you can make whatever changes you want by either modifying settings already in the configmap or adding new ones. For more information about these options, see Prepare the Web Gateway configuration files.

When data nodes are mirrored, the IKO generates a separate configmap for the nodes in each mirror, for example **`my-iriscluster-data-0`** for node 1 of a mirrored sharded cluster, **`my-iriscluster-data-1`** for the second data node, and so on. These corresponds to the node naming scheme for mirrored configurations as described in Understand IrisCluster node naming.

## 17.6 imagePullSecrets: Provide a secret containing image pull information

```
  imagePullSecrets:
  - name: name
  - ...
```

**Optional**

The `imagePullSecrets` field specifies one or more Kubernetes secrets containing the URL of the registry from which images to be pulled and the credentials required for access.

Kubernetes secrets let you securely and flexibly store and manage sensitive information such as credentials that you want to pass to Kubernetes. To enable Kubernetes to download an image from a secure registry, 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.

Create another Kubernetes secret, like the one you created for the IKO image pull information, for the InterSystems IRIS image and others you intend to deploy, such as arbiter, InterSystems Web Gateway, and so on. For example, if Kubernetes will be pulling these images from the InterSystems Container Registry (ICR) as described in Obtain the IKO image, you would use a command like the one shown below. 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==********** 
```

Finally, update the `name` field in `imagePullSecrets` with the name of the secret you created; if you did not create one, do not specify a value for this field. If you create multiple image pull secrets, you can specify multiple secret names in the `imagePullSecrets` field.

If you include multiple secrets in `imagePullSecrets` because the images specified in multiple `image` fields in the definition are in different registries, Kubernetes uses the registry URL in each `image` field to choose the corresponding image pull secret. If you specify just one secret, it is the default image pull secret for all image pulls in the definition.

## 17.7 storageClassName: Create a default class for persistent storage

```
storageClassName: name
```

**Optional**

Specifies the Kubernetes storage class to use by default for the predefined persistent volumes provided by the IKO and for any custom persistent volumes you request, as described in Plan persistent volumes.

Storage class is one characteristic of a persistent volume that can be specified in the persistent volume claim that describes it. Storage classes provide a way for administrators to describe the types of storage offered in a given Kubernetes environment. For example, different classes (sometimes called "profiles" on other provisioning and deployment platforms) might map to quality-of-service levels, backup policies, specialized hardware, or arbitrary policies germane to the deployments involved. You can specify an existing storage class, or a new storage class you define in Kubernetes prior to deploying the cluster, as the default for all persistent volumes in your IrisCluster by putting its name in the `StorageClassName` field. If you do not specify a default, the default storage characteristics are specific to the Kubernetes platform you are using; consult the platform documentation for details.

You can also override the default storage class (whether set by you in the `storageClassName` field or platform-specific) for one or more of the predefined persistent volumes deployed with data, compute, webgateway, sam, and iam nodes by adding the `storageClassName` field to the appropriate volume definition within the node type definition. To do this for custom volumes you add to your deployment, you can include the `storageClassName` field either persistent volume claims you define using the `volumeClaimTemplates` field or in persistent volumes you add to the `data` or `compute` node definitions using the `volumeMounts` field.

**Important:** Any storage class you define must include Kubernetes setting `volumeBindingMode: WaitForFirstConsumer` for correct operation of the IKO.

## 17.8 updateStrategy: Select a Kubernetes update strategy

```
updateStrategy:
  type: RollingUpdate
```

**Optional**

Specifies the update strategy that Kubernetes uses to update the stateful sets in the deployment. The value can be either `RollingUpdate` (the default) or `OnDelete`. This setting can be overridden by using the `updateStrategy` field within a node type definition in the `topology` section to specify the update strategy for that type of node only.

## 17.9 volumeClaimTemplates: Define persistent storage volumes

```
volumeClaimTemplates:
- metadata:
    name: nameN
  spec:
    accessModes:
    - {ReadWriteOnce|ReadOnlyMany|ReadWriteMany}
    resources:
      requests:
        storage: size
    storageClassName: nameN
- ...
```

**Optional**

Defines one or more persistent volume claims to be used to create persistent storage volumes (see Plan persistent volumes), which can be mounted on `data` or `compute` nodes using the volumeMounts field. Any field from the Kubernetes persistent volume claim spec can be included; for example, as shown, you can use `resources` to override the default volume size, and `storageClassName` to override the deployment's default storage class. However, the only required settings is the name of the template in the `metadata` section, as all of the others that must be defined have defaults.

When you do specify volume size in the `storage` field, it can be in any unit between kilobytes and exabytes

## 17.10 volumes: Request ephemeral storage volumes

```
volumes:
- name: nameN
  type:
    typeName: name
- ...
```

**Optional**

Specifies one or more ephemeral storage volumes, which are mounted on `data` or `compute` nodes using the `volumeMounts` field. An ephemeral volume stores data that is used by an application but (unlike a persistent volume) does not need to persist across restarts, for example read-only input such as configuration data and licenses. (For an example of using ephemeral volumes to provide such data, see `licenseKeySercret`.) Because ephemeral volumes are created and deleted along with pods, pods using them can be stopped and restarted without having to maintain access to a particular persistent volume. There are several types of ephemeral volume, including configMap, secret, emptyDir, and others.

**Important:** Volume names provided must be unique across the `volumeClaimTeplates` and `volumes` fields; if there are volumes of either type with duplicate names, an error will occur when Kubernetes attempts to mount one of them as specified in the `volumeMounts` field.

## 17.11 serviceTemplate: Create external IP addresses for the cluster

```
serviceTemplate:
  spec:
    type: LoadBalancer
    externalTrafficPolicy: Local
    ports:
    - name: name
      port: portnumber
    - ...
```

**Optional**

Provides access to the IrisCluster deployment by defining one or more Kubernetes services, which expose an application running on a set of pods by assigning an external IP address. In addition, lets you open specified ports in all `data` node containers, `compute` node containers, or both.

At a minimum, the `serviceTemplate` field creates an external IP address assigned to one of the pods in the first stateful set managing `data` node pods, which is used to connect requests with the InterSystems IRIS superserver and web server ports (1972 and 52773, respectively). For example, the external IP can be used to access the Management Portal on the exposed data node.

Other services and external IP addresses created represent the first pod in the first stateful set managing `webgateway` pods, `sam` pods, and `iam` pods, if these nodes are included in the IrisCluster. For information about connecting to all of these services through their external IP addresses, see Connect to the IrisCluster.

**Important:** For information about distributing application (query) connections to the recommended nodes within the deployment (and about IrisCluster node names), see Prepare the Web Gateway configuration file.

The `data` node exposed by this service is selected as follows.

- If the deployment is a sharded cluster and the data nodes are not mirrored, the service exposes data node 1, that is, the node named *clustername*`-data-0`. If the data nodes are mirrored, the service exposes the deployed primary of data node 1, *clustername*`-data-0-0`, but because all connections to sharded cluster data nodes are mirror aware, the IP address always represents the current primary.
- If the deployment is a distributed cache cluster or standalone instance, the service exposes *clustername*`-data-0` if the single data node is not mirrored. If the data node is mirrored:
  - If an `arbiter` node is included in the deployment, the service is mirror aware and exposes the current primary, for example either *clustername*`-data-0-0` or *clustername*`-data-0-1`.
  - If an arbiter node is not included in the deployment, the service is not mirror aware and therefore exposes the deployed primary, *clustername*`-data-0-0`, which means that the IP address will not follow failover. For this reason, InterSystems strongly recommends that you always deploy an arbiter with a mirrored distributed cache cluster or standalone instance.

    **Note:** `Compute` node (application server) connections to the mirrored data node in a distributed cache cluster are always mirror aware.

To open ports on data and/or compute nodes, provide port names with the following prefixes:

| Node type | Prefix |
|---|---|
| `data` | `data-` |
| `compute` | `compute-` |
| `data` and `compute` | `iris-` |

For example:

```
serviceTemplate:
  spec:
    type: LoadBalancer
    externalTrafficPolicy: Local
    ports:
    - name: data-iscagent
      port: 2188
    - name: compute-ssh
      port: 22
    - name: iris-python
      port: 8081
```

**Note:** The values `LoadBalancer` for `type:` and `Local` for `externalTrafficPolicy` (as shown) are required.

## 17.12 tls: Configure TLS security

```
tls:
  common:
    volume_source:
      volume_source_spec:
      ...
  mirror:
    volume_source:
      volume_source_spec:
      ...
  webgateway:
    volume_source:
      volume_source_spec:
      ...
  superserver:
    volume_source:
      volume_source_spec:
      ...
  ecp:
    volume_source:
      volume_source_spec:
      ...
  iam:
    volume_source:
      volume_source_spec:
      ...
```

#### Optional but recommended

Configures TLS security, which InterSystems strongly recommends for all deployments, for applicable types of connection to and within the IrisCluster, as follows:

- `webgateway`: To the management pages (used to configure the Web Gateway) of each webgateway node.
- `superserver`: To the superserver port of the exposed data node and between each webgateway node and the data and compute nodes in its server access configuration.
- `mirror`: Within data node mirrors.
- `ecp`: Between compute nodes and data nodes (in a distributed cache cluster, the application servers and the data server).
- `iam`: To the IAM portal.
- `common`: All of the above.

**Important:** Web gateway connections are secured as described above only if you use the configuration files generated by the IKO; InterSystems recommends you verify the TLS configurations of the entries in the server access configuration after deployment. Be sure to see Changing the Web Gateway's default password for more information about securing server access configuration connections.

If both `common` and an individual type are specified, the individual specification overrides the common specification for that type of connection.

Each specification consists of an existing volume source for the required certificate files; in the following example, the specified volume sources for both `common` and `webgateway` are secrets (which could be Kubernetes TLS secrets), but the certificate files for data node mirrors are obtained from a CSI volume.

```
tls:
  common:
    secret:
      secretName: common-certs
  mirror:
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "my-provider"
  webgateway:
    secret:
      secretName: webgateway-certs
```

Valid names for certificate files are tls.crt, tls.key, and ca.pem. All three are required for every connection type but `iam`, which does not require the ca.pem file. If one of the required files is missing from the source, deployment halts with an error identifying the missing file. You can use the **kubectl create secret** command to :include the required certificate files in a secret, as shown in the following:

```
kubectl create secret generic common-certs --from-file=tls.crt --from-file=tls.key --from-file=ca.pem
```

## 17.13 topology: Define the cluster nodes

```
topology:
  data:
    ...
  compute:
    ...
  arbiter:
    ...
  webgateway:
    ...
  sam:
    ...
  iam:
```

#### Required

Specifies the details of the each type of cluster node to be deployed. As described in Define the IrisCluster topology, the IrisCluster must have one or more data nodes, so the `data` section, defining the data nodes, is required; all other node types are optional.

**Important:** Be sure to review information about securing connections to and within the IrisCluster in tls: Configure TLS security.

# 17.14 data: Define sharded cluster data nodes or distributed cache/standalone data server

```
data:
  image: registry/repository/image:tag
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
    - ...
  podTemplate:
    core.PodTemplateSpec
  shards: N
  mirrored: {true|false}
  storage{DB|WIJ|Journal1|Journal2}:
    resources:
      requests:
        storage: spec
    mountPath
  volumeMounts:
  - mountPath: pathN
    name: volumeClaimTemplateN or volumeN
  - ...
  containers:
  - name: containerName
    image: registry/repository/image:tag
    field: value
    ...
  - ...
  compatibilityVersion: iris-version
```

**Required**

The `data` section defines the IrisCluster's data nodes, of which there must be at least one. Only the `image` field is required within the `data` section.

## 17.14.1 image:

```
image: containers.intersystems.com/intersystems/iris:2022.2.0.205.0
```

**Required**

The `image:` field specifies the URL (registry, repository, image name, 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). The registry credentials in the secret specified by the imagePullSecrets field are used for access to the registry.

The `image` fields in other node definitions are also required, and are used in the same way. The `sam` node definition has four optional additional image fields for third-party images.

**Important:** Two default settings in InterSystems IRIS containers deployed from the secure **`iris-lockeddown`** image (see Locked Down InterSystems IRIS Container in *Running InterSystems Products in Containers*) prevent InterSystems System Alerting and Monitoring (SAM), if it is part of the deployment, from gaining access to the instance, as follows:

- The instance's private web server is disabled.
- The allowed authentication method for the **/api/monitor** web application is set to password authentication rather than unauthenticated.

When a sam node is included in the deployment, the IKO ensures that it has access to the InterSystems IRIS instances in all containers deployed from the secure **`iris-lockeddown`** image by enabling the private web server and changing the authentication method for the **/api/monitor** web application to unauthenticated.

Another consequence of the private web server being disabled is that the Management Portal becomes inaccessible. To enable the Management Portal for data nodes deployed from this image, so you can use it to connect to data node 1 (or the single data server) as described in Connect to the IrisCluster, add the setting `webserver=1` to the data.cpf configuration merge file described in configSource: Create configuration files and a configmap for them.

## 17.14.2 updateStrategy:

```
updateStrategy:
  type: {RollingUpdate|OnDelete}
```

**Optional**

Overrides the top level updateStrategy setting to specify the Kubernetes update strategy used for the stateful sets representing this node type only. The value can be either `RollingUpdate` (the default) or `OnDelete`.

## 17.14.3 preferredZones:

```
preferredZones:
  - zoneN
  - ...
```

**Optional**

Specifies the zone or zones in which data nodes should be deployed, and is typically used as follows:

- If `mirrored` is set to `true` and at least two zones are specified, Kubernetes is discouraged (but not prevented) from deploying both members of a failover pair in the same zone, which maximizes the chances that at least one is available, and is therefore the best practice for high availability. Bear the following mind, however:
  - Deploying the members of a failover pair in separate zones is likely to slightly increase latency in the synchronous communication between them.
  - Specifying multiple zones for the data nodes means that all of the primaries might not be deployed in the same zone, resulting in slightly increased latency in communication between the data nodes.
  - Specifying multiple zones for data nodes generally makes it impossible to guarantee that nodes of other types (compute, Web Gateway, SAM, IAM) are in the same zone as all of the data node primaries at any given time regardless of your use of `preferredZones` in their definitions, increasing latency in those connections as well.

  Under most circumstances these interzone latency effects will be negligible, 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 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 minimize 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.
- The value of the `preferredZones:` field in the `compute` definition, if included, should ensure that the `compute` nodes are deployed in the same zone or zones as the data nodes to minimize latency (see Plan Compute Nodes in the *Scalability Guide*).

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

## 17.14.4 podTemplate:

```
podTemplate:
  spec:
    args:
```

**Optional**

Specifies overrides and additions to the default pod template applied to the data node pods (and to the pods of other node types in their respective definitions). Because containers are run only within a pod on Kubernetes, a pod template can specify the fields that define numerous Kubernetes entities, including the fields defining a container, which makes it useful for many purposes. Several examples of using containers fields are provided in the following:

- Apply one or more labels to the pod:

```
podTemplate:
  metadata:
    labels:
      name: value
       ...
```

- Prevent InterSystems IRIS from starting up with the container.

  You can use the `args` field in the pod template to specify options to the InterSystems IRIS entrypoint application, **iris-main**. For example, if there is something wrong with the InterSystems IRIS configuration which prevents startup from succeeding, **iris-main** exits, causing the pod to go into a restart loop, which makes it difficult or impossible to diagnose the problem. You can prevent the instance from starting by adding the **iris-main** option **--up false** as follows:

```
podTemplate:
  spec:
    args:
    - --up
    - "false"
```

  When you do this, the readiness probe will not be satisfied, and the deployment will be paused indefinitely:

```
$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
my-IrisCluster-data-0   0/1     Running   0          32s
```

  After addressing the problem, you can do one of the following:

  – Manually start the instance with a command like the following:

```
$ kubectl exec -it my-IrisCluster-data-0 -- iris start IRIS
```

  – Remove the `podTemplate` override and redeploy the pod.

- Adjust the termination grace period and specify lifecycle hooks.

  When a pod is terminated because the IrisCluster configuration is updated (see Modify the IrisCluster) or due to an explicit action (such as a **kubectl delete** command), the amount of time it has to complete termination is determined by the `terminationGracePeriodSeconds` setting. If this time is too short, InterSystems IRIS may not have enough time to shut down before the pod is terminated, which results in upgrades failing because the instance did not shut down properly. For this reason, the default value of `terminationGracePeriodSeconds` is 3600 seconds (as opposed to the Kubernetes default of 30 seconds).

  The pod template can also specify the `postStart` and `preStop` lifecycle hooks, which are executed immediately after a container is created and immediately before a container is terminated due to an API request or management event, respectively. (Execution of the latter is limited by the `terminationGracePeriodSeconds` setting.)

  The following example sets the value of `terminationGracePeriodSeconds` and specifies the command executed by the InterSystems IRIS entrypoint application, **iris-main**, when it receives SIGTERM as the `preStop` hook:

```
podTemplate:
  spec:
    terminationGracePeriodSeconds: 600
    lifecycle:
      preStop:
        exec:
          command: ["/bin/sh", "-c", "iris stop IRIS quietly"]
```

- Override the default liveness probe and readiness probe.

  If you wanted to replace the default liveness probe and readiness probe for data nodes (or another node type), you could specify something like the following in the applicable pod template:

```
podTemplate:
  spec:
    livenessProbe:
      exec:
        command:
        - /bin/true
    readinessProbe:
      httpGet:
        path: /csp/user/cache_status.cxw
        port: 52773
```

- Limit the number of CPU cores a `data` or `compute` node container can use in order to remain within the limits of the InterSystems IRIS license in use; limit the memory containers can use for efficient resource distribution.

  You can use the `resources` field in the pod template to both request and limit resources to be used by the containers in the pod. In the following example, the pod starts with 256 MB of memory and 2 CPU cores, and is limited to 2 GB of memory and 2 CPU cores:

```
podTemplate:
  spec:
    resources:
      requests:
        memory: "256Mi"
        cpu: "2"
      limits:
        memory: "2Gi"
        cpu: "2"
```

- Allocate huge pages on cluster nodes.

  You can also use `resources` to allocate huge pages on InterSystems IRIS or other nodes:

```
podTemplate:
  spec:
    resources:
      requests:
        memory: "2Gi"
      limits:
        memory: "2Gi"
        hugepages-2Mi: "2Gi"
```

**Important:** The default pod template for `data` node and `compute` node pods includes a security context with the correct settings for InterSystems IRIS containers. Bear in mind that adding security context fields to these or any pod template could cause errors. For example, as described in Security for InterSystems IRIS Containers, the InterSystems IRIS, arbiter, and Web Gateway instances are installed by and must run as user **`irisowner`**/**`51773`** in their respective containers, so adding the security context field `runAsUser` with any other value would cause deployment of these containers to fail. If you want to specify a security context for any of the pods in your IrisCluster deployment, be sure to consult the documentation for the product involved and review all security mechanisms before doing so.

## 17.14.5 compatibilityVersion

Allows backwards/forwards compatibility between certain versions of InterSystems IRIS and IKO. In IKO version 3.5, the default value for this field is `2022.2.0`; other possible values include `2022.3.0` and higher.

## 17.14.6 shards:

```
shards: N
```

**Optional**

Specifies the number of data nodes to be deployed as a sharded cluster. If the `shards` field is omitted, a single standalone instance of InterSystems IRIS is deployed, optionally as the data server in a distributed cache cluster; for more information, see Define the IrisCluster topology.

Data nodes can be added to the deployed cluster by increasing this setting and reapplying the definition (as described in Modifying the IrisCluster), but the setting cannot be decreased .

## 17.14.7 mirrored:

```
mirrored: {true|false}
```

**Optional**

Determines whether the data nodes in the deployment are mirrored.

If the value of `mirrored` is `true`, two mirrored instances, consisting of primary and backup failover members, are deployed by default 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, creating a mirrored sharded cluster with four data nodes. If `mirrored:` is `true` when `shards` is omitted, two mirrored instances are deployed as a standalone InterSystems IRIS instance, which can optionally be the mirrored data server of a distributed cache cluster; for details, see Define the IrisCluster topology.

The default for `mirrored` is `false`.

**Important:** Connections to mirrored data nodes within a sharded cluster are mirror aware — that is, they are made to whichever member is currently primary, making failover transparent. This is also true of application server connections to a mirrored data server in a distributed cache cluster, and of connections to a standalone mirror through a webgateway node. If you deploy a standalone mirror that receives external connections directly, however, these connections are *not* mirror-aware unless you include an `arbiter` node in the deployment. Without an arbiter, connections are always to *clustername*`-data-0-1`, even if the mirror has failed over to *clustername*`-data-0-2`.

A deployed cluster *cannot* be changed from unmirrored to mirrored, or mirrored to unmirrored, by changing this setting and reapplying the definition (as described in Modifying the IrisCluster).

## 17.14.8 mirrorMap:

```
mirrorMap: primary,backup,drasync, ... drasync
```

As noted above, data nodes or standalone instances deployed when `mirrored` is `true` consist by default of two instances, a primary and a backup. You can add one or more disaster recover (DR) async members using the `mirrorMap` field. The number of DR asyncs depends on how many times you include `drasync` in the field's value. For example, if you include `mirrorMap: primary,backup,drasync`, each mirror data node, or the mirrored standalone instance, consists of a primary, a backup, and a DR async; if you include `mirrorMap: primary,backup,drasync,drasync,drasync`, each mirror consist includes the two failover members and three DR asyncs. The maximum number of mirror members is 16, therefore you can add up to 14 DR asyncs to each mirror.

The default for `mirrorMap` is `primary,backup`.

When using this field, bear the following in mind:

- The `mirrorMap` field takes effect only when `mirrored` is `true` and `compatibilityVersion` is `2022.3.0` or higher.
- Async members are not configured as data shards; they are just normal instances that belong to the mirror.
- Be mindful of the ratio of mirror members to availability zones; for example, two availability zones and three mirror members might result in the primaries of two data nodes being deployed in different availability zones. For more information about the effects of deploying in multiple availability zones, see `preferredZones`.

## 17.14.9 mirrorName:

```
mirrorName: basename
```

**Optional**

Specify a custom basename for any defined mirrors.

By default, mirrors created by IKO currently use the following naming scheme:

```
IRISMIRROR1
IRISMIRROR2
IRISMIRROR3
...
```

You can specify a different basename by providing a value for the `mirrorName` field. For example, `mirrorName: MYMIRROR` would result in the following mirror names:

```
MYMIRROR1
MYMIRROR2
MYMIRROR3
...
```

## 17.14.10 storage*:

```
storage{DB|WIJ|Journal1|Journal2}:
  resources:
    requests:
      storage: size
  storageClassName: storage-class-name
```

**Optional**

Specify custom characteristics, such as size or storage class, for one or more of the four predefined persistent volumes deployed with each data node, as follows:

- `storageDB` — The volume on which data is stored, including durable %SYS data.

- `storageWIJ` — The volume containing the WIJ directory.
- `storageJournal1` — The volume containing the primary journal directory.
- `storageJournal2` — The volume containing the alternate journal directory.

These predefined volumes are mounted in /irissys inside the container, and are 2 GB by default. In addition to specifying a size override, you can include any other field from the Kubernetes persistent volume claim spec to override default characteristics. For example, you can add `storageClassName` to override the deployment's default storage class, as shown.

When including a size override, the value of the `storage` field can be specified in any unit between kilobytes and exabytes. The amount of data storage to be mounted on sharded cluster data nodes is determined during the cluster planning process and should include a comfortable margin for the future growth of your workload. In addition to overriding the sizes of the predefined volumes, you can use additional persistent volumes defined in the `volumeClaimTemplates` field and specified in the `volumeMounts` field to ensure that sufficient storage is available to each data node.

The same four `storage*` fields can be used to modify the same predefined volumes in the `compute` node definition in the same ways. These volumes are also 2 GB by default. The data storage for sharded cluster compute nodes or distributed cache cluster application servers should be kept to a bare minimum to conserve resources, as these nodes do not store application data, so size overrides of these volumes on compute nodes may be desirable.

Storage volumes on data and/or compute nodes in a deployed cluster can be expanded, provided the storage class specified by `storageClassName` is defined to allow volume expansion. To expand a volume, increase the `storage:` value in the definition and reapply it (as described in Modifying the IrisCluster). The command **kubectl get pvc** shows the cluster's current storage volumes sizes, while **kubectl get iriscluster** shows the volume sizes currently in the cluster definition (whether applied or not), and **kubectl get sts** shows the values from the definition applied when the cluster was created, without reflecting subsequent changes.

In the `webgateway` node definition you can override the characteristics of the single predefined volume, `storageDB`. The default size of this volume is 32 MB. Predefined volumes are also deployed with `sam` and `iam` nodes, and you can specify overrides for these in their respective definitions.

## 17.14.11 volumeMounts:

```
volumeMounts:
- mountPath: pathN
  name: volumeClaimTemplateN or volumeN
- ...
```

**Optional**

Specifies one or more persistent storage volumes, as defined in the `volumeClaimTemplates` field, and/or ephemeral storage volumes, as specified in the `volumes` field, to be deployed with each data node, in addition to the predefined persistent volumes. Each volume is defined by the name of one of the volume claim templates or volumes and a `mountPath`, which is a direct reference to a location in the container's filesystem, visible to the InterSystems IRIS instance, on which to mount the volume.

The `volumeMounts` field can also be used to specify additional (typically ephemeral) volumes in the `compute` and webgateway node definitions.

## 17.14.12 containers:

```
containers:
- name: containerName
  image: registry/repository/image:tag
  field1: value
  ...
- ...
```

**Optional**

Specifies one or more containers to run in the pod alongside each data node (so-called *sidecar containers*). To define each sidecar container, create an entry in the array of `core.Container`; only `name` and `image` are required.

The `containers` field can also be used to specify sidecar containers in the `compute` definition. This field takes effect only when `compatibilityVersion` is `2022.3.0` or higher.

# 17.15 compute: Define sharded cluster compute nodes or application servers

```
compute:
  image: containers.intersystems.com/intersystems/iris:2022.2.0.205.0
[common InterSystems IRIS node fields, optional for all node types]
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
    - ...
  podTemplate:
    core.PodTemplateSpec
[end common InterSystems IRIS node fields]
  replicas: N
  storage{DB|WIJ|Journal1|Journal2}:
    resources:
      requests:
        storage: spec
  volumeMounts:
  - mountPath: pathN
    name: volumeClaimTemplateN or volumeN
  - ...
  containers:
  - name: containerName
    image: registry/repository/image:tag
    field: value
    ...
  - ...
```

**Optional**

The `compute` section defines the IrisCluster's compute nodes. As described in Define the IrisCluster topology, if the IrisCluster will be deployed as a sharded cluster (because the `shards` field is included in the `data` section), you can use `compute` to add compute nodes to the cluster, but if a single `data` node will be deployed as a standalone instance because `shards` is omitted, defining compute nodes adds application servers, creating a distributed cache cluster.

If the `compute` section is included, only the `image` and `replicas` fields are required. For information about the remaining `compute` fields, see the `data` section.

## 17.15.1 image

```
image: containers.intersystems.com/intersystems/iris:2022.2.0.205.0
```

**Required in optional compute: section**

Compute nodes are deployed from the same InterSystems IRIS image as `data` nodes; an example is shown.

## 17.15.2 replicas:

```
        replicas: N
```

**Required in optional compute: section**

Specifies the number of identical compute nodes to deploy. In a sharded cluster, this should be a multiple of the number of data nodes specified by `shards` (for more information, see Plan Compute Nodes in the *Scalability Guide*).

Compute nodes can be added to or removed from the deployed IrisCluster by changing this setting and reapplying the definition (as described in Modifying the IrisCluster).

The `replicas` field also appears in the `webgateway` definition, where it specifies the number of InterSystems Web Gateway (web server) nodes to deploy, and the `sam` and `iam` definitions, where it is used to remove an existing sam or iam node from a deployment.

**Note:** In place of the `replica` field, you can use a Kubernetes HorizontalPodAutoscaler to automatically increase or decrease the number of compute nodes based on factors such as CPU load. The following example includes all of the required fields:

```
compute:
  image: containers.intersystems.com/intersystems/iris:2023.1.0L.156.0
  compatibilityVersion: "2023.1.0"
  hpa:
    spec:
      minReplicas: 2
      maxReplicas: 4
      scaleTargetRef:
        kind: StatefulSet
        name: acme-compute
        apiVersion: app/v1
```

The `compatibilityVersion` field must be set to 2023.1.or later to use the `hpa` field. IKO updates the contents of `scaleTargetRef` to match the cluster, but the section must be present (with any values) in order for Kubernetes to validate the definition.

## 17.16 arbiter: Define arbiter for mirrored data nodes

```
arbiter
  image: containers.intersystems.com/intersystems/arbiter:2022.2.0.205.0
[common InterSystems IRIS node fields, optional for all node types]
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
  podTemplate:
    core.PodTemplateSpec
[end common InterSystems IRIS node fields]
```

#### Optional

The `arbiter` section defines an arbiter node to be deployed with a mirrored sharded cluster or data server. For general information about the `arbiter` fields, see the `data` section, noting the following arbiter-specific information:

- The arbiter is deployed from an InterSystems **`arbiter`** image, an example of which is shown in the `image` field.
- Use the `preferredZones` field in the `arbiter` definition to deploy the arbiter in a separate zone from the `data` node mirror members to optimize mirror availability.

**Important:** InterSystems strongly recommends including an arbiter node in a mirrored distributed cache cluster or standalone instance deployment; for more information, see serviceTemplate: Create external IP addresses for the cluster.

**Note:** To add an arbiter to existing mirrored data nodes deployed without one, add an `arbiter` entry to the `topology` section and reapply the IrisCluster definition, as described in Modifying the IrisCluster.

# 17.17 webgateway: Define web server nodes

```
webgateway:
  image: containers.intersystems.com/intersystems/webgateway:2022.2.0.205.0
[common InterSystems IRIS node fields, optional for all node types]
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
  podTemplate:
    core.PodTemplateSpec
[end common InterSystems IRIS node fields]
  storageDB:
    resources:
      requests:
        storage: spec
  volumeMounts:
  - mountPath: pathN
    name: volumeClaimTemplateN or volumeN
  - ...
  type: {apache|apache-lockeddown|nginx}
  replicas: N
  applicationPaths:
  - pathN
  - ...
```

**Optional**

The `webgateway` section defines the web server (webgateway) nodes to be deployed. Each webgateway node includes the InterSystems Web Gateway, which provides the communications layer between the hosting web server and InterSystems IRIS for web applications, and an Apache or Nginx web server. Multiple webgateway nodes can de deployed as a web server tier for a sharded cluster, a distributed cache cluster, or a standalone instance, mirrored or unmirrored.

If the `webgateway` section is included, only the `image` and `replicas` fields are required unless `image` specifies a **`webgateway-nginx`** or **`webgateway-lockeddown`** image, in which case you must also include `type: nginx` or `type: apache-lockeddown`, respectively. For information about the remaining `webgateway` fields not discussed here, see the `data` section, noting the following `webgateway`-specific information:

- Among the predefined volumes, you can override the size of the `storageDB` data volume, which is used for storing configuration and log files; its default size is 32 MB.
- Depending on your circumstances, you may want to use `preferredZones` to locate your web server tier relative to the data and compute nodes they connect to.

As previously described, the `serviceTemplate` field creates one or more Kubernetes services to expose the IrisCluster to the network through external IP addresses. If you include webgateway nodes, an external IP address representing the first pod in the first stateful set managing Web Gateway pods is created. This IP address can be used with the URL listed in Connect to the IrisCluster to connect to the Web Gateway management pages on that node and review the Web Gateway configuration, and can then propagate any changes you make to the other webgateway nodes as described in Updating the configuration of multiple Web Gateways.

**Important:** Although webgateway nodes are optional, deploying one or more is the simplest way to follow best practices in distributing application connections to a sharded or distributed cache cluster and is therefore recommended. For more information, see Prepare the Web Gateway configuration file.

At this time, the IKO does not automatically expose all of the webgateway nodes to the network. To enable load balancing of application connections across the web server tier, you can manually define a service exposing the nodes, which on some platforms can include a load balancer.

Be sure to review information about securing webgateway connections in Changing the Web Gateway's default password and tls: Configure TLS security.)

### 17.17.1 image

```
image: containers.intersystems.com/intersystems/iris:2022.2.0.205.0
```

**Required in optional webgateway: section**

Use one of the following InterSystems images to deploy webgateway nodes:

- **`webgateway`** (as shown in the example) — Deploys an InterSystems Web Gateway instance and an Apache web server.
- **`webgateway-lockeddown`** — To meet the strictest security requirements, deploys a nonroot Web Gateway instance installed with locked-down security and a nonroot Apache web server configured to use port 52773 instead of the standard port 80.
- **`webgateway-nginx`** — Deploys a Web Gateway instance and an Nginx web server.

For information about these images and the differences between them, see Using the InterSystems Web Gateway Container in *Running InterSystems Products in Containers*.

### 17.17.2 type:

```
type: {apache|apache-lockeddown|nginx}
```

**Optional**

Specifies deployment of an Apache web server, a locked down Apache web server, or an Nginx web server (as described for the previous field); the default is `apache`.

**Important:** The image specified in the `image` field must match the value of the `type` field, or the webgateway nodes will either fail to deploy or be inaccessible.

### 17.17.3 replicas:

```
replicas: N
```

**Required in optional webgateway: section**

Specifies the number of identical webgateway nodes to deploy.

You can add webgateway nodes to, or remove them from, the deployed IrisCluster by changing this setting and reapplying the definition (as described in Modifying the IrisCluster).

The `replicas` field also appears in the `compute` section, where it specifies the number of compute nodes to deploy.

### 17.17.4 applicationPaths:

```
applicationPaths:
  - pathN
  - ...
```

**Optional**

Provides a list of application paths to configure in the Web Gateway. Application paths should not have a trailing slash, and the path /csp is reserved.

### 17.17.5 alternativeServers:

```
alternativeServers: {FailOver|LoadBalancing}
```

**Optional**

Selects the method by which the Web Gateway on each node determines which InterSystems IRIS server (that is, which `data` node) in its server access configuration to connect to (see Load Balancing and Failover Between Multiple InterSystems IRIS Server Instances in the *Web Gateway Guide*). Possible values are `FailOver` and `LoadBalancing`, with a default of `LoadBalancing`.

## 17.18 sam: Deploy System Alerting and Monitoring

```
sam:
  image: containers.intersystems.com/intersystems/sam:2.0.0.123
[common InterSystems IRIS node fields, optional for all node types]
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
    - ...
  podTemplate:
    core.PodTemplateSpec
[end common InterSystems IRIS node fields]
  replicas: {1|0}
  storage{SAM|storageGrafana}:
    resources:
      requests:
        storage: spec
  image[AlertManager|Grafana|Nginx|Prometheus]: registry/repository/image:tag
  loginSecret:
    name: secret-name
```

**Optional**

The `sam` section deploys System Alerting and Monitoring (SAM), a cluster monitoring solution for InterSystems IRIS data platform, along with the selected InterSystems IRIS topology. For general information about the `sam` fields, see the `data` section, noting the following SAM-specific information:

- SAM is deployed from an InterSystems **`sam`** image, an example of which is shown in the `image` field.
- The `replicas` field can be set to 1 (deploy a sam node) or 0 (do not deploy). The default is 1, so this field is not required when deploying a sam node, and if you do not want to deploy SAM you can simply omit the `sam` definition. The `replicas` field can be used, however, to remove a deployed `sam` node by adding `replicas: 0` to the definition and then reapplying the definition as described in Modify the IrisCluster.
- The `storageSam` storage override field differs in name from those in the `data` and other sections, but functions in the same way, providing the ability to override the size of the predefined volume for the SAM Manager container. (For information about the predefined SAM volume, see SAM Component Breakdown in the *System Alerting and Monitoring Guide*.)
- The `imageAlertManager`, `imageGrafana`, `imageNginx`, and `imagePrometheus` fields allow you to override the IKO's default image specs for these third-party images.
- If you want to the SAM configuration to be automatically updated when you change the number of nodes in your cluster (as described in Modify the IrisCluster), you must use the `loginSecret` field to specify a Kubernetes secret containing the password needed to authenticate calls to SAM's REST API, allowing the IKO to modify the configuration at runtime. If you do not include this field, changes to the number of nodes result in SAM reporting errors trying to reach instances that no longer exist and being unable to monitor instances that were added.

**Important:** When deploying `sam` nodes using IKO version 3.5, deploy from the SAM version 2.0 image (as shown above) or later. For information about InterSystems container images, see Using the InterSystems Container Registry.

## 17.19 iam: Deploy InterSystems API Manager

```
iam:
  image: containers.intersystems.com/intersystems/iam:2.3.3.2-1
[common InterSystems IRIS node fields, optional for all node types]
  updateStrategy:
    type: {RollingUpdate|OnDelete}
  preferredZones:
    - zoneN
    - ...
  podTemplate:
    core.PodTemplateSpec
[end common InterSystems IRIS node fieldss]
  replicas: {1|0}
  storagePostgres:
    resources:
      requests:
        storage: spec
  imagePostgres: registry/repository/image:tag
```

**Optional**

The `iam` section deploys the InterSystems API Manager (IAM), which enables you to monitor and control traffic to and from web-based APIs, along with the selected InterSystems IRIS topology. For general information about the `iam` fields, see the `data` section, noting the following IAM-specific information:

- IAM is deployed from an InterSystems **`iam`** image, an example of which is shown in the `image` field.
- The `replicas` field can be set to 1 (deploy an iam node) or 0 (do not deploy). The default is 1, so this field is not required when deploying an iam node, and if you do not want to deploy IAM you can simply omit the `iam` definition. The `replicas` field can be used, however, to remove a deployed `iam` node by adding `replicas: 0` to the definition and then reapplying the definition as described in Modify the IrisCluster.
- The `storagepostgres` storage override field in the `iam` section differs in name from those in the `data` section, but functions in the same way, providing the ability to override the size of the predefined volume for the IAM container.
- The `imagePostgress` field allows you to override the IKO's default image spec for this third-party images.

# 18 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
```

**Note:** In Kubernetes, a pod's readiness probe is used to tell you when the services in deployed containers are fully started and ready for operation. This is indicated by the status `Running`, as shown in the preceding. Whether your cluster uses the default readiness probe or you specified another in the cluster definition, as described for the podTemplate field, it is normal for the probe to fail the first two or three times it runs after a container starts up. As long as the readiness probe succeeds soon thereafter, and all of the pods have the status `Running`, these initial failures do not represent a problem and can safely be ignored.

# 19 Connect to the IrisCluster

As previously described, the `serviceTemplate` field creates one or more Kubernetes services to expose the IrisCluster to the network through external IP addresses. For example, the service for data node 1 (or the data server or the standalone instance), which is always created, is used to connect to the superserver and web server ports (1972 and 52773, respectively) of the InterSystems IRIS instance running on that node for data ingestion and other purposes.

**Important:** The IKO does not create a service that distributes application (query) connections to the recommended nodes within the deployment; for information about doing so, see Prepare the Web Gateway configuration file.

For example, to load the cluster's Management Portal in your browser, get the data node 1 IP address by listing the services representing the IrisCluster, as follows:

```
$ kubectl get svc
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
my-IrisCluster-Webgateway LoadBalancer  10.35.245.9  35.196.145.177  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
```

Other services and external IP addresses created, if applicable, represent the first pod in the first stateful set managing Web Gateway pods, SAM pods, and IAM pods, if these nodes are included in the IrisCluster. The URLs (including ports) for these connections are as follows:

| Service | URL including port |
|---|---|
| Web Gateway, `type`={`nginx`\|*apache*} | `http://external-ip:80/csp/bin/Systems/Module.cxw` |
| Web Gateway, `type=apache-lockeddown` | `http://external-ip:52773/csp/bin/Systems/Module.cxw` |
| SAM | `http://external-ip:8080/api/sam/app/index.csp` |
| IAM | `http://external-ip:8002/overview` |

# 20 Troubleshoot 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.

The `podTemplate` field can be useful in exploring deployment and startup errors; examples are provided in that section.

- **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.

**Note:** In Kubernetes, a pod's readiness probe is used to tell you when the services in deployed containers are fully started and ready for operation.Whether your cluster uses the default readiness probe or you specified another in the cluster definition, as described for the podTemplate field, it is normal for the probe to fail the first two or three times it runs after a container starts up. As long as the readiness probe succeeds soon thereafter, these initial failures do not represent a problem and can safely be ignored.

# 21 Modify the IrisCluster

Generally speaking, you can make changes to your IrisCluster by modifying the definition file (using a change management system to keep track of your modifications) and repeating the **kubectl apply** command shown in Deploy the IrisCluster. For example, you can modify the cluster by:

- Adding `data` nodes to a sharded cluster or changing the number of `compute` nodes in a sharded cluster or distributed cache cluster.
- Expanding `storage volumes` on data and/or compute nodes.
- Adding an `arbiter` to a `mirrored` cluster or standalone instance without one or removing the one you originally specified.
- Adding or removing a `sam` or `iam` node.
- Changing the number of `webgateway` nodes.

You cannot reduce the number of data nodes, diminish storage volumes, or change a deployment's mirror status; other changes may produce unanticipated issues.

To change the configurations of data, compute, and webgateway nodes without reapplying the definition, you can edit the configmap; for details see Modify configmaps to reconfigure nodes.

# 22 Upgrade the IrisCluster

To upgrade the InterSystems IRIS instances in an IrisCluster from version 2021.1 to version 2021.2 or later, follow these steps:

1. Upgrade the IKO from version 3.1 to version 3.3 using the steps from Download the IKO archive and upload the extracted contents to Kubernetes through Install the IKO, but in the last step substituting the **helm upgrade** command for **helm install**, for example:

   ```
   helm upgrade intersystems iris_operator-3.3.0.118.0/chart/iris-operator
   ```

2. Update the cluster definition file by replacing the InterSystems IRIS 2021.1 images with 2021.2 images in the image fields in the `data` and (if applicable) `compute` sections.
3. Apply the change, for example:

```
kubectl apply -f my-IrisCluster-definition.yaml
```

**Important:** If more than a few seconds elapse between step 1 and step 3, the data and compute node pods may enter an error or **`CrashLoopBackOff`** state because they rebooted before the image change took effect. This would result in a permissions issue visible in the pod's logs. If this happens, delete the affected pods using **kubectl delete pod** *pod-name*.

# 23 Remove the IrisCluster

To fully remove the cluster, you must issue a **kubectl** command to delete not only the cluster, but also the persistent volumes/volume claims associated with it, for example:

```
kubectl delete -f my-IrisCluster-definition.yaml
```

To uninstall the IKO, issue the following command:

```
helm uninstall intersystems
```

You can also fully remove the IrisCluster and the IKO by unprovisioning the Kubernetes cluster on which they are deployed.