Using the InterSystems Kubernetes Operator (Version 3.6)

2024-05-06

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

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 3.6)

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 IKO?

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, 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 Kubernetes 1.24 and later versions. Use with the most recent compatible version is recommended.

7 Deploy a Kubernetes container cluster

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, upload it to Kubernetes

Obtain the IKO archive file, for example iris_operator-3.6.7.100-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-3.6.7.100) 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:3.6.7.100`**, 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-3.6.7.100/image/iris_operator-3.6.7.100-docker.tgz
fd6fa224ea91: Loading layer [==================================================>] 3.031MB/3.031MB

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

Loaded image: intersystems/iris-operator:3.6.7.100
$ docker images
REPOSITORY                  TAG         IMAGE ID      CREATED       SIZE
intersystems/iris-operator  3.6.7.100  9a3756aed423  3 months ago  77.3MB
$ docker tag intersystems/iris-operator:3.6.7.100 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 an image pull secret

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 currently supported for use with the IKO, for example:

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

  # Operator Environment Variables
  useFQDN: true
```

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-3.6.7.100/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 distribution on deployment, deployed web server nodes do provide such a mechanism, since the Web Gateway can distribute requests received from the web server across multiple InterSystems IRIS instances as defined in its server access profiles (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 names

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 profiles 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-1myCluster-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.
- 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.

Included in the IKO /Samples directory is a file called iris-sample.yaml containing a very simple IrisCluster definition deploying a stand-alone InterSystems IRIS Community Edition instance and creating a service that exposes it as an external IP address. You may find it helpful to review this definition, and you can deploy it without or with modifications as a test.

Note: Because a Community Edition image is used, the web server port for Management Portal access through the exposed IP address would be 52733, not 80.

17.1 Review the IrisCluster custom resource definition (CRD)

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

```
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
      - ...
      irisDatabases:
      - name: irisDatabaseN
        directory: path
        mirrored: {true|false}
        ecp: {true|false}
        seed: path
        logicalOnly: {true|false}
      - ...
      irisNamespaces:
      - name: irisNamespaceN
        routines: database-name
        globals: database-name
        interop: {true|false}
```

```
      - ...
      containers:
      - name: containerName
        image: registry/repository/image:tag
        field: value
        ...
      - ...
      webgateway:
        image: registry/repository/image:tag
        type: {apache|apache-lockeddown|nginx}
        applicationPaths:
        - pathN
        - ...
        loginSecret:
          name: secret-name
    compute:
      image: registry/repository/image:tag
     see common InterSystems IRIS node fields in data definition
      replicas: N
      ephemeral:
      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
        ...
      - ...
      webgateway:
        image: registry/repository/image:tag
        type: {apache|apache-lockeddown|nginx}
        applicationPaths:
        - pathN
        - ...
        loginSecret:
          name: secret-name
    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}
      ephemeral:
      storageDB:
        resources:
          requests:
            storage: spec
        storageClassName:
      volumeMounts:
      - mountPath: pathN
        name: volumeClaimTemplateN or volumeN
      - ...
      loginSecret:
        name: secret-name
    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 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.

17.3 licenseKeySecret: Provide a secret containing the InterSystems IRIS license key

```
  licenseKeySecret:
    name: secret-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: data-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.

17.4 configSource: Create configuration files and generate a configmap from them

```
configSource:
  name: configmap-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.
- A Web Gateway configuration file called CSP.ini to be installed in the `webgateway` container on 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

17.4.1 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 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
```

17.4.2 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 predefined account, which by default provides management access to the Web Gateway in `webgateway` nodes and `webgateway` sidecar containers. For more information, see the following section.

17.4.3 Prepare the Web Gateway configuration files

As explained in Web Access Using the Web Gateway Container in *Running InterSystems Products in Containers*, the InterSystems Web Gateway container, which includes both a web server and a Web Gateway instance, has two different purposes in containerized deployments, as follows:

- To provide a packaged web server component for distributing application connections across multiple InterSystems IRIS nodes, as in a distributed cache or sharded cluster.
- To serve as a dedicated web server for a containerized InterSystems IRIS instance’s built-in web applications, including the web-based Management Portal.

Your IrisCluster can include one or more `webgateway` nodes for the first purpose, and a sidecar Web Gateway container in each `data` and/or `compute` node pod for the second.

A Web Gateway’s configuration, which is contained in its CSP.ini file, includes server access profiles specifying the InterSystems IRIS servers (instances) with which it interacts and application access profiles specifying which application requests are directed to which of these instances. For a `webgateway` node interacting with multiple InterSystems IRIS nodes, applications may be evenly or differentially distributed across them. However, a dedicated (sidecar) Web Gateway container has only the server access profile for the instance it is dedicated to and is intended for built-in applications on that instance only, so all web URLs are directed to the server identified by that profile. For these reasons, the CSP.ini files for the two types of Web Gateway have many differences.

The IKO generates the CSP.ini file for all dedicated Web Gateway containers. For `webgateway` nodes, you have two options for specifying a CSP.ini file in your configmap to be automatically installed on every `webgateway` node deployed: you can let the IKO generate the CSP.ini file, or you can provide your own. In using the latter option, 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.

Note: You can also supply your own CSP.conf file containing the Web Gateway's Apache or Nginx-specific configuration, but this is not 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.

Creating your own CSP.ini file for web server webgateway nodes is practical if your IrisCluster definition includes limited number of `data`, `compute`, and `webgateway` nodes. In more complex deployments, InterSystems recommends using the IKO-generated file because of the following advantages:

- The IKO-generated CSP.ini file automatically distributes application connections according to best practices, as described in Populate the server access profiles in the CSP.ini file. This can be done in a user-provided CSP.ini, but creating such a file can be time-consuming and error-prone. Assuming you do not supply your own CSP.conf file, IKO also generates one that configures the needed application paths on the web server.
- Using the IKO-generated file allows you to include a CSP-merge.ini file in your configmap, which provides the easiest and most effective means of updating the configurations of multiple webgateway nodes in a single operation, as described in Synchronize reconfiguration of multiple webgateway nodes.
- The IKO-generated CSP.ini file automatically secures connections between `webgateway` nodes and `data` and `compute` nodes with TLS if you so specify in the `tls` section of the IrisCluster definition, as well as configuring the Web Gateway's credentials for authenticating to the InterSystems IRIS instances specified in its server access profiles according to the information you provide in the `loginSecret` field.

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:

- Populate the server access profiles in the CSP.ini file
- Secure Web Gateway management access
- Update and coordinate server authentication credentials
- Synchronize reonfiguration of multiple webgateway nodes

Populate the server access profiles 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.) To take advantage of this, use the IKO-generated CSP.ini, or in a user-provided file ensure that the server access profiles, which specify the InterSystems IRIS instances to which the Web Gateway directs incoming connections from the web server, are populated according to the relevant best practice for distributing application connections, as follows:

- If the deployment is a sharded cluster, create server access profiles for all `data` nodes and `compute` nodes (if any).
- If the deployment is a distributed cache cluster, create profiles for all `compute` (application server) nodes.
- If the deployment is a standalone instance, its profile should be the one in the configuration.

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

As noted in Understand IrisCluster node naming, it is possible to populate the server access profiles 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 profiles 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**

Important: InterSystems recommends the use of TLS to secure 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.

Secure Web Gateway management access

Access to the Web Gateway's management pages can be secured in three ways: by specifying the required authentication credentials, by restricting the IP addresses from which the pages can be accessed, and by disabling access entirely. InterSystems recommends that you review and update these settings as follows:

On `webgateway` nodes:

- If you provide your own CSP.ini file, you can make your desired changes to all of these settings in the `[SYSTEM]` section of the file:
  - If you want to disable management access altogether, which means the configuration can be changed only by modifying the CSP.ini file in your configmap and using the **kubectl replace** command to propagate your changes to the `webgateway` nodes, set the SM_Forms parameter to Disabled. Otherwise, address the next two items.
  - Specify authentication credentials of your choice using the `Username` and `Password` parameters. If the password is in plain text, it will be base64-encoded and given a prefix of `[[[` when each Web Gateway starts up; if the prefix is already present, the password will be used as-is.
  - The `System_Manager`setting, in the same section, specifies a list of IP addresses corresponding to machines permitted to access the management pages; if it is omitted, access is limited to the local host. Set this according to your expectations regarding the need for access to the management pages.

- If you use the IKO-generated CSP.ini file, the specified authentication credentials are **`CSPSystem`**/**`SYS`** (with the password encrypted). Because this is one of the predefined accounts on InterSystems IRIS and is thus well known, InterSystems recommends changing these credentials. You can modify them as well as the `System_Manager` setting (which by default allows all IP addresses), or disable access to the management pages using the `SM_Forms` parameter, by including a CSP-merge.ini file in your configmap, with contents like one of the following:

```
[SYSTEM]
SM_Forms=Disabled

[SYSTEM]
Username=username
Password=password
System_Manager=IPaddress1,IPaddress2,...
```

In `webgateway` sidecar containers:

- The settings described above, including the authentication credentials. are the same in the IKO-generated CSP.ini for `webgateway` sidecars as in the IKO-generated CSP.ini for `webgateway` nodes. If you use the latter, you can modify the file `webgateway` sidecar configuration file along with the `webgateway` node file by including a CSP-merge.ini file in your configmap as described above.
- If you do not use the IKO-generated CSP.ini for `webgateway` nodes, you can modify the CSP.ini in each sidecar using the **kubectl exec** command, as described in Synchronize reconfiguration of multiple webgateway nodes.

Update and coordinate server authentication credentials

To interact with an InterSystems IRIS instance, the Web Gateway must authenticate to it, which is why each Web Gateway server access profile specifies the username and password of an account on the InterSystems IRIS instance it specifies to be used for this purpose.

To ensure that all Web Gateways can authenticate to InterSystems IRIS as needed out of the box, the IKO-generated CSP.ini files for the `webgateway` nodes and the `webgateway` sidecar containers both specify the credentials **`_SYSTEM`**/**`SYS`** for this purpose in the server access profiles for the `data` nodes and `compute` nodes to which they will distribute application connections and the InterSystems IRIS container they are dedicated to, respectively. This is one of the well-known predefined accounts for which the password should be changed during or immediately after deployment (for details, see Authentication and Passwords in *Running InterSystems Products in Containers).*

For both for the `webgateway` nodes and the `webgateway` sidecar containers, you can use the `loginSecret` field to change the password, or both the username and password, in all server access profiles in both IKO-generated CSP.ini files. This must be the same for both he `webgateway` nodes and the `webgateway` sidecar containers due to the need for coordination with the InterSystems IRIS nodes (see below). If you provide your own CSP.ini for the `webgateway` nodes, the `loginSecret` field has no effect; instead, specify a secure account that does not use the default password in each remote server access profile ([*`server-name`*] section). This must be the same for all server profiles as well as matching the credentials you supply in a `loginSecret` for the `webgateway` sidecar containers,

Once you have determined the server access credentials to be specified using the `loginSecret` field (and possibly your own CSP.ini file), you must ensure that these credentials are valid on the `data` and `compute` nodes with which the `webgateway` nodes and sidecars will be interacting. To do this, create the necessary entry in the data.cpf, compute.cpf, or common.cpf you include in your configmap, as described in Prepare the configuration merge files, as follows:

- If you are including only a password in your login secrets, thereby retaining the default username **`_SYSTEM`** (or you provided this username in your own CSP.ini), you need use only the `PasswordHash` parameter in the common.cpf file you provide, as described in Set the default InterSystems IRIS password, to change all default passwords, including the one for **`_SYSTEM`**, on all of the targeted InterSystems IRIS nodes to the encrypted value you supply.
- If you specified a new username as well, you must use the `CreateUser` action parameter to create the account on all of the targeted instances; you can find an example of using this parameter to create an account at deployment in Create, Modify and Delete Security Objects in *Automating Configuration of InterSystems IRIS with Configuration Merge*.

Synchronize reconfiguration of multiple webgateway nodes

As described in Connect to the IrisCluster, the `serviceTemplate` definition creates one or more Kubernetes services that expose a pod in the first StatefulSet of certain node types 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, where *`external-ip`* is the IP address exposed by the relevant service. (For nodes deployed from the **`webgateway-lockeddown`** image, specify port 52773 instead of 80 in the URL.)

If you have deployed multiple `webgateway` nodes, you can make configuration changes across all of them by modifying a file in your configmap and using the **kubectl replace** command to propagate your changes to the `webgateway` nodes. In this regard, an advantage of using the IKO-generated CSP.ini file is that you can include in your configmap a file called CSP-merge.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, both when you first deploy your IrisCluster and when you subsequently reconfigure the `webgateway` nodes. If you supply your own CSP.ini, you must make your updates in the entire file when you edit the configmap. Using the IKO-generated file and CSP-merge.ini lets you make updates more efficiently and (assuming you track of them using change control) more easily review the update history.

Important: When you use the CSP-merge.ini file to modify the IKO-generated CSP.ini file at deployment or later, the settings you include in CSP-merge.ini are also applied to the CSP.ini file in the dedicated (sidecar) `webgateway` containers. Generally speaking, due to the fact that the sidecar `webgateway` file contains only the LOCAL server profile and only built-in InterSystems IRIS paths in its application profiles, while the `webgateway` node file does not contain the former and typically not the latter, changes intended for the `webgateway` nodes should not affect the dedicated (sidecar) webgateway containers. However it is important to consider this possibility when updating the IKO-generated `webgateway` node CSP.ini using the CSP-merge.ini file.

If you deploy with the IKO-generated CSP.ini file, when you modify the IrisCluster by updating the definition to change the number of `data` or `compute` node or add or remove `sam` or `iam` node and then reapplying the cluster definition, a new CSP.ini file, reflecting the changes (including in the distribution of application connections) is automatically generated and installed on the `webgateway` nodes; any changes you made to the previous file using CSP-merge.ini will have to be made again if still desired, either when reapplying the modified definition or later reconfiguring the nodes as described above.

Be sure to review information about securing Web Gateway connections in Secure Web Gateway management access ,Update and coordinate server authentication credentials, and tls: Configure TLS security.

Note: If you supply your own the CSP.ini file, you can modify this file on individual `webgateway` nodes in the following ways:

- Define a service similar to the one created by the `serviceTemplate` for each `webgateway` pod that exposes it through an external IP address.
- Directly modify the CSP.ini file on each pod by using the **kubectl exec** command to open a shell and edit the file within the container.

17.4.4 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
```

17.4.5 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.

17.4.6 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 Synchronize reconfiguration of multiple webgateway nodes, 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); in either case you can make whatever changes you want by either modifying settings already in the configmap or adding new ones.

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 correspond to the node naming scheme for mirrored configurations as described in Understand IrisCluster node naming.

17.5 imagePullSecrets: Provide a secret containing image pull information

```
imagePullSecrets:
- name: pullsecret-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 those for the other nodes you intend to deploy, such as `arbiter`, `webgateway`, 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.

17.6 storageClassName: Create a default class for persistent storage

```
storageClassName: storageclass-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.

17.7 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.

17.8 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

17.9 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.

17.10 serviceTemplate: Create external IP addresses for the cluster

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

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 it an external IP address. In addition, lets you open specified ports in all `data` node pods, all `compute` node pods, or both.

At a minimum, the `serviceTemplate` field creates an external IP address assigned to a pod in the first statefulSet managing `data` nodes, which enables applications to connect InterSystems IRIS through the superserver port (1972), and (by way of its sidecar `webgateway` container, if defined) provides access to the instance's web-based Management Portal (and other built-in web applications) through the web server port (80), as well as the instance's superserver port (1972).

Note: If the sidecar `webgateway` container was deployed from the **`webgateway-lockeddown`** image, specify port 52773 instead of 80 in the URL.

Similar services and external IP addresses are created to expose the first pod in the first statefulSet 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 how `webgateway` nodes distribute application connections to the `data` and `compute` nodes in a deployment (and about IrisCluster node names), see Prepare the Web Gateway configuration files.

The `data` node exposed by this service is data node 1 in a sharded cluster and the single data node in a distributed cache cluster or stand-alone instance. The name of the exposed node varies as follows:

- If the data nodes are not `mirrored`, the service exposes the node named *clustername*`-data-0`.
- If the data nodes are `mirrored` and an `arbiter` node is defined, the service always exposes the current primary of the data node, *clustername*`-data-0-0` (the deployed primary) or *clustername*`-data-0-1` (the deployed backup).
- If the data nodes are `mirrored` and no `arbiter` node is defined, the service can expose only the deployed primary, *clustername*`-data-0-0`. The consequences of this depend on the configuration:
  - All connections to sharded cluster data nodes are mirror aware, so for a sharded cluster the IP address always represents the current primary, even if the current primary is *clustername*`-data-0-1`.
  - If the deployment is a distributed cache cluster or standalone instance, the service is not mirror aware; it can expose only the deployed primary, *clustername*`-data-0-0`, and the IP address will not follow failover. For this among other reasons, 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.

You can use the `ports` field to open additional ports on all `data` nodes, all `compute` nodes, or both. The former adds to the ports available through the default data node 1 service described above, but both make ports available through any services you may additionally define for `data` node or `compute` node pods. The port names you provide must begin 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
```

For information about other possible values for `type` and `externalTrafficPolicy` when creating Kubernetes services, see the Kubernetes services documentation.

17.11 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 first `data` node and between each `webgateway` node and the `data` and `compute` nodes in its server access profiles.
- `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 server access profiles after deployment. Be sure to see Secure Web Gateway management access for more information about securing server 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.

17.12 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.

17.13 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
  compatibilityVersion: iris-version
  shards: N
  mirrored: {true|false}
  storage{DB|WIJ|Journal1|Journal2}:
    resources:
      requests:
        storage: spec
    mountPath
  volumeMounts:
  - mountPath: pathN
    name: volumeClaimTemplateN or volumeN
  - ...
  containers:
  - name: container-name
    image: registry/repository/image:tag
    field: value
    ...
  - ...
  webgateway:
    image: registry/repository/image:tag
    type: {apache|apache-lockeddown|nginx}
    applicationPaths:
    - pathN
    - ...
    loginSecret:
      name: secret-name
```

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.

17.13.1 image:

```
image: containers.intersystems.com/intersystems/irislatest-em
```

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.

17.13.2 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`.

17.13.3 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.

17.13.4 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: 1972
  ```

- 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"
```

- Override the mount point for durable storage.

  By default, the `storageDB` predefined persistent volume, which includes durable %SYS data, is mounted as /usr/irissys in the container's file system. If you want to mount the volume at another location, you can override the default by changing the durable %SYS location and the mount point for `storageDB`, as follows:

```
data:
  podTemplate:
    spec:
      env:
      - name: ISC_DATA_DIRECTORY
        value: "/irissys/data2/"
  storageDB:
    mountPath: "/irissys/data2"
```

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 in *Running InterSystems Products in 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.

17.13.5 compatibilityVersion

Required

Indicates the version of InterSystems IRIS being deployed in the IrisCluster; its value must match the version of the InterSystems IRIS image specified in the image field in the data and (optionally) compute node definitions.

17.13.6 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.

If you have deployed from a definition that does not contain this field — that is, either a distributed cache cluster or a standalone instance — you can convert the IrisCluster to a sharded cluster by:

- Adding the `shards` field.
- If necessary, modifying the number of `compute` nodes to make it a multiple of the value you specified for `shards` by changing the value of the replicas field, or by adding a `compute` node definition to the `topology` section if there is none.
- Making any other needed or desired modifications, such as adding a web server tier through the `webgateway` definition, changing the persistent storage definitions for the `data` nodes, adding mirroring, and so on.

17.13.7 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).

17.13.8 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`.

17.13.9 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
...
```

17.13.10 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), mounted by default as /usr/irissys in the container's file system.
- `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.

17.13.11 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.

17.13.12 irisDatabases:

Optional

```
irisDatabases:
- name: irisDatabaseN
  directory: path
  mirrored: {true|false}
  ecp: {true|false}
  seed: path
  logicalOnly: {true|false}
- ...
```

Specifies one or more databases to be created on persistent volumes, whether on the predefined `storageDB` predefined persistent volume (the default) or a volume you defined under `volumeMounts`, and mounted on the `data` node instance. Except for `name`, the following fields are optional:

- `name`

  The name of the database, which must be unique.

- `directory`

  The location in the container file system or on a persistent volume in which to create the database. If omitted, the default is /usr/irissys/mgr/*database-name*, unless the mount point for the `storageDB` persistent volume has been changed, as illustrated in the last example in the `podTemplate` section, in which case the new mount point replaces /usr/irissys.

- `mirrored`

  If true and the IrisCluster is mirrored (the value of `data`/`mirrored` is true), the database is created as a mirrored database in the mirror specified by the `mirrorName` field.

- `ecp`

  If true, the database is configured for access by ECP clients; this is necessary for the database to be accessible to `compute` nodes in a sharded or distributed cache cluster.

- `seed`

  The location in the container file system or on a persistent volume of an existing, nonmirrored database from which the new database is to be cloned to the container filesystem or persistent volume location specified by `directory`. For example, if `name` is **databaseCA**, `directory` is **/usr/irissys/mgr**, and `seed` is **/my-persistent-volume-C/databaseA**, **`databaseA`** is cloned to /usr/irissys/mgr/databaseCA.

- `logicalOnly`

  If true, the existing database specified by `seed` is to be mounted at the location specified by `directory`, rather than being cloned.

17.13.13 irisNamespaces:

Optional

```
irisNamespaces:
- name: irisNamespaceN
  routines: database-name
  globals: database-name
  interop: {true|false}
- ...
```

Specifies one or more namespaces to be created on the `data` node. The `name` and `globals` fields are required, the other are optional:

- `name`

  The name of the namespace, which must be unique.

- `routines`

  The name of the default routines database for the namespace.

- `globals`

  The name of the default globals database for the namespace.

- `interop`

If true, the namespace will be interoperability-enabled.

17.13.14 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.

17.13.15 webgateway: Sidecar Web Gateway containers

```
webgateway:
  image: registry/repository/image:tag
  type: {apache|apache-lockeddown|nginx}
  applicationPaths:
  - pathN
  - ...
loginSecret:
  name: secret-name
```

Optional, but recommended for InterSystems IRIS Management Portal access

The InterSystems Web Gateway passes HTTP requests for a web server to InterSystems IRIS instances, including both application connections and access to each instance's web-based Management Portal. In containerized deployments, however, a single Web Gateway instance can provide access to just one InterSystems IRIS instance's Management Portal. The simplest means of addressing this issue, as described in detail in Web Access Using the Web Gateway Container in *Running InterSystems Products in Containers*, is to deploy a dedicated Web Gateway container with every InterSystems IRIS container for the sole purpose of providing access to the Management Portal and other built-in web applications. In an IrisCluster, dedicated Web Gateway containers used for this purpose are defined separately from the web server nodes defined in the `webgateway` section of `topology`.

On Kubernetes platforms, the standard way to address the need for companion containers is to add so-called sidecar containers to the relevant pods, and the `containers` field described above is used for this purpose generally. But given the required Web Gateway and web server configuration, a dedicated Web Gateway sidecar would be difficult to configure and maintain, and would not take advantage of the automatic configuration available for web server nodes. The optional `webgateway` field in the `data` definition allows you to easily add a dedicated Web Gateway container to every `data` node pod as a sidecar. In addition to being configured to provide Management Portal access, the Web Gateway sidecar provides web access to `iam` and `sam` nodes, if defined.

The fields in the sidecar `webgateway` definition are a subset of those in the web server node (`webgateway`) definition. Specific usage is as follows:

- You must specify a value for the `image` field and can optionally specify values for the `type` and `applicationPaths` fields. To serve the Management Portal of the accompanying InterSystems IRIS, you must define /csp/sys as an application path. You should also configure /csp/broker, which provides access to files shared by several built-in system applications.
- If you are specifying a secret containing a password and optionally username in the `loginSecret` field of the `webgateway` definition, specify the same secret with the same field here. (For more information, see Update and coordinate server authentication credentials.)

- The replicas and ephemeral fields do not apply to the sidecar webgateway; there is always one, and it is always ephemeral.

17.14 compute: Define sharded cluster compute nodes or application servers

```
compute:
  image: registry/repository/image:tag
[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
  ephemeral: {true|false}
  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
    ...
  - ...
  webgateway:
    image: registry/repository/image:tag
    type: {apache|apache-lockeddown|nginx}
    applicationPaths:
    - pathN
    - ...
    loginSecret:
      name: secret-name
```

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.

17.14.1 image

```
image: containers.intersystems.com/intersystems/iris:latest-em
```

Required

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

Note: The image tags shown in this document are examples only. Please go to the InterSystems Container Registry (ICR) to browse current repositories and tags.

17.14.2 replicas:

```
replicas: N
```

Optional

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*).

You can add compute nodes to, or remove them from, the deployed IrisCluster by modifying the configmap or by changing this setting and reapplying the definition.

The `replicas` field also appears in the `webgateway` definition, where it specifies the number of web server nodes to deploy and is required, 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:latest-em
  compatibilityVersion: "2023.2.0"
  hpa:
    spec:
      minReplicas: 2
      maxReplicas: 4
      scaleTargetRef:
        kind: StatefulSet
        name: acme-compute
        apiVersion: app/v2
```

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.

17.14.3 ephemeral

```
emphemeral: {true|false}
```

Optional

When set to true, only ephemeral (not persistent) volumes are created. If any storage is required, the local disk is used. Default is false.

Use this option if you want the cluster's compute nodes to come up more quickly, do not plan to install any custom applications, and do not want to preserve any state on the compute nodes. Can also be used in the `webgateway` definition on the same basis.

17.15 arbiter: Define arbiter for mirrored data nodes

```
arbiter
  image: containers.intersystems.com/intersystems/arbiter:latest
[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.

17.16 webgateway: Define web server nodes

```
webgateway:
  image: registry/repository/image:tag
[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
  ephemeral: {true|false}
  applicationPaths:
  - pathN
  - ...
  alternativeServers: {FailOver|LoadBalancing}
  loginSecret:
    name: secret-name
```

Optional, but recommended for automatic distribution of application connections

The `webgateway` section defines the web server nodes to be deployed. The container on each web server 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 web server nodes can be 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: appache-lockeddown`, respectively. For information about the remaining `webgateway` fields not discussed here, see the `data` and `compute` section, noting the following `webgateway`-specific information:

- The replicas field is required in the `webgateway` definition, although it is optional in `compute`.
- 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.

Important: Although `webgateway` nodes are optional, defining one or more and using the IKO-generated Web Gateway configuration file enables your IrisCluster to automatically distribute application connections to a sharded or distributed cache cluster in accordance with best practices, and is therefore recommended. For more information, see Prepare the Web Gateway configuration files.

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 Secure Web Gateway management access, Update and coordinate server authentication credentials, and tls: Configure TLS security.

Note: To deploy sidecar Web Gateway containers in `data` and `compute` node pods to provide access to the Management Portal and other built-in web applications, include the `webgateway` field in those definitions.

17.16.1 image

```
image: containers.intersystems.com/intersystems/webgateway:latest
```

Required

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 Web Access Using the Web Gateway Container in *Running InterSystems Products in Containers*.

17.16.2 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.

17.16.3 replicas:

```
replicas: N
```

Required

Specifies the number of identical webgateway nodes to deploy.

You can add webgateway nodes to, or remove them from, the deployed IrisCluster by modifying the configmap or by changing this setting and reapplying the definition.

The `replicas` field also appears in the `compute` section, where it specifies the number of compute nodes to deploy, and the `sam` and `iam` definitions, where it is used to remove an existing sam or iam node from a deployment.

17.16.4 ephemeral:

```
emphemeral: {true|false}
```

Optional

When set to true, only ephemeral (not persistent) volumes are created. If any storage is required, the local disk is used. Default is false.

Use this option if you want the cluster's web server nodes to come up more quickly, do not plan to install any custom applications, and do not want to preserve any state on the web server nodes. Can also be used in the `compute` definition on the same basis.

17.16.5 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.

17.16.6 alternativeServers:

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

Optional

Selects the method by which the Web Gateway on each node determines which of the InterSystems IRIS servers — that is, which `data` or (optionally) `compute` node — in its server access profiles 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`.

17.16.7 loginSecret:

```
loginSecret:
  name: secret-name
```

Optional, but recommended for effective security

The `loginSecret` field specifies a Kubernetes secret containing a `password` and optional `username` entry with which the Web Gateway authenticates to the InterSystems IRIS instances specified in its server access profiles. For information about the use of these credentials to connect to InterSystems IRIS and their role in Web Gateway configuration, see Update and coordinate server authentication credentials.

Create a generic secret for this purpose, as in this example:

```
kubectl create secret generic webgateway-secret --from-literal=username=wg-auth
--from-literal=password=[[[U1lT8g7
```

If the password is in plain text, it is base64-encoded and given the prefix of [[[to indicate this encoding; if the prefix is already present, the password is used as is.

You must also supply a `loginSecret` as described above when specifying a dedicated `webgateway` sidecar container in the `data` and/or `compute` definitions, which due to the need to coordinate with InterSystems IRIS should be the same as the one provided here. (You can also specify a similar `loginSecret` for a different purpose in the `sam` definition.)

17.17 sam: Deploy System Alerting and Monitoring

```
sam:
  image: registry/repository/image:tagcontainers.intersystems.com/intersystems/sam:2.0.1.181
[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 or later, 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.

17.18 iam: Deploy InterSystems API Manager

```
iam:
  image: containers.intersystems.com/intersystems/iam:3.2.1.0-4
[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

The `serviceTemplate` field creates one or more Kubernetes services to expose the IrisCluster to the network through external IP addresses. For example, a service is always created for the first pod in the first stateful set managing `data` nodes, which represents data node 1 in a sharded cluster, the data server in a distributed cache cluster, or a stand-alone instance, and is used to connect to the InterSystems IRIS instance running on that node for web applications, application connections, data ingestion and other purposes. You can connect to the superserver port (1972) by default, and to the web server port (80) if you included a `webgateway` sidecar in your `data` node definition. (If you deployed the sidecar from a **`webgateway-lockeddown`** image, the web server port is 52773.) To get the IP address for the node, list 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,80:31887/TCP
 46m
my-IrisCluster-Webgateway  LoadBalancer   10.35.245.9   35.196.145.177   80:31887/TCP
 46m
```

You can then make connections to the node at the displayed external IP address, as illustrated in the following:

- To open the cluster's Management Portal in your browser, use the following URL:

  http://*external-ip*:80/csp/sys/UtilHome.csp

- You can use a similar URL to interact with any of the built-in web applications on the first `data` node. For example, for the InterSystems Public Key Infrastructure you would use

  http://*external-ip*:80/isc/pki/PKI.CAServer.cls

- To make connections to the superserver port of InterSystems IRIS on the first `data` node, use *external-ip*:1972. For example, when connecting through the JDBC driver to load data into a sharded cluster, or for relational data access from a Java application, you would form the JDBC connection string as follows:

  ```
  jdbc:IRIS://external-ip:1972/namespace
  ```

If you opened additional ports using the `ports` field in your `serviceTemplate` definitions, these will be available as well.

You can also define a similar service for the pod representing any other data or compute node if you wish to access its Management Portal or another web application through its `webgateway` sidecar, or to connect to its superserver port or any additional open ports you specified using the `ports` field.

Other services and external IP addresses created by default, if the applicable nodes are included in the IrisCluster, represent the first pod in the first stateful set managing `webgateway` nodes, `sam` nodes, and `iam` nodes. 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 IKO Version

To upgrade the Kubernetes cluster's IKO version, use the steps from Download the IKO archive and upload the extracted contents to Kubernetes through Install the IKO, but in the last step substitute the **helm upgrade** command for **helm install**, for example:

```
helm upgrade intersystems iris_operator-3.6.7.100/chart/iris-operator
```

23 Upgrade the IrisCluster

To upgrade the version of InterSystems IRIS deployed on the cluster's `data` and `compute` nodes do the following:

1. Change the value of the compatibilityVersion field in the `data` and (if applicable) `compute` sections of the cluster definition file to the new version of InterSystems IRIS that you will be deploying.
2. Update the InterSystems IRIS images specified in the `image` field of the `data` and `compute` sections definition file to the new version.
3. Apply the changes with a command like the following:

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

If you are using the **`latest-em`** or **`latest-cd`** tag, you do not need to replace the InterSystems IRIS image specification before using **kubectl apply**; the latest versions of these images will be downloaded and deployed in the IrisCluster.

24 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.