InterSystems
IRIS Data Platform

InterSystems Cloud Manager Guide

Version 2019.1
2019-05-13

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

InterSystems Cloud Manager Guide
InterSystems IRIS Data Platform Version 2019.1 2019-05-13

Table of Contents

List of Figures

List of Tables

About This Document

This document describes the use of InterSystems Cloud Manager (ICM) to deploy InterSystems IRIS Data Platform™ configurations in public and private clouds and on preexisting physical and virtual clusters.

This book contains the following chapters:

- ICM Overview
- Essential ICM Elements
- Using ICM
- ICM Reference

This book also contains the following appendixes:

- Containerless Deployment
- Sharing ICM Deployments
- Scripting with ICM
- Using ICM with Custom and Third-Party Containers
- Deploying on a Preexisting Cluster

1
ICM Overview

This chapter explains what InterSystems Cloud Manager (ICM) does, how it works, and how it can help you deploy InterSystems IRIS Data Platform configurations on cloud, virtual, and physical infrastructure.

- Benefits of ICM
- The ICM Application Lifecycle

Note: For a brief introduction to ICM including a hands-on exploration, see First Look: InterSystems Cloud Manager.

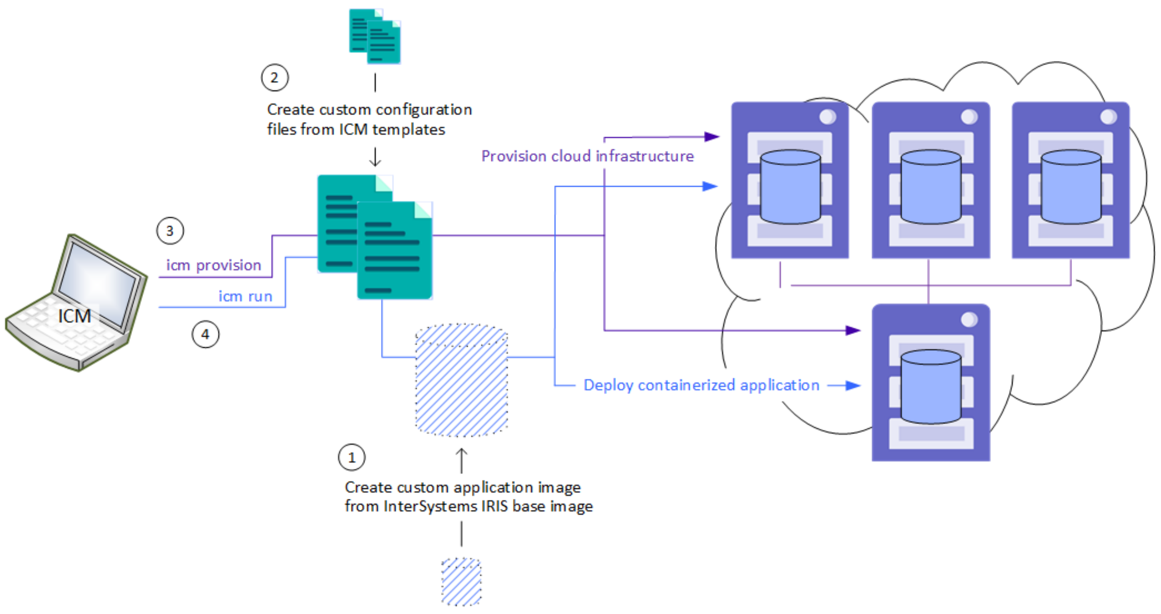
1.1 Benefits of ICM

InterSystems Cloud Manager (ICM) provides you with a simple, intuitive way to provision cloud infrastructure and deploy services on it. ICM is designed to bring you the benefits of infrastructure as code (IaC), immutable infrastructure, and containerized deployment of your InterSystems IRIS™-based applications, *without* requiring you to make major investments in new technology and the attendant training and trial-and-error configuration and management.

ICM makes it easy to provision and deploy the desired InterSystems IRIS configuration on Infrastructure as a Service (IaaS) public cloud platforms, including Google, Amazon, and Azure. Define what you want in plain text configuration files and use the simple command line interface to direct ICM; ICM does the rest, including provisioning your cloud infrastructure with the widely-used Terraform IaC tool and deploying your InterSystems IRIS-based applications on that infrastructure in Docker containers.

ICM codifies APIs into declarative configuration files that can be shared among team members like code, edited, reviewed, and versioned. By executing Terraform commands as specified by these files, ICM enables you to safely and predictably create, change, and improve production infrastructure on an ongoing basis.

Figure 1–1: ICM Makes It Easy


2
Create custom configuration files from ICM templates
Provision cloud infrastructure
3
icm provision
icm run
4
ICM
Deploy containerized application
1
Create custom application image from InterSystems IRIS base image

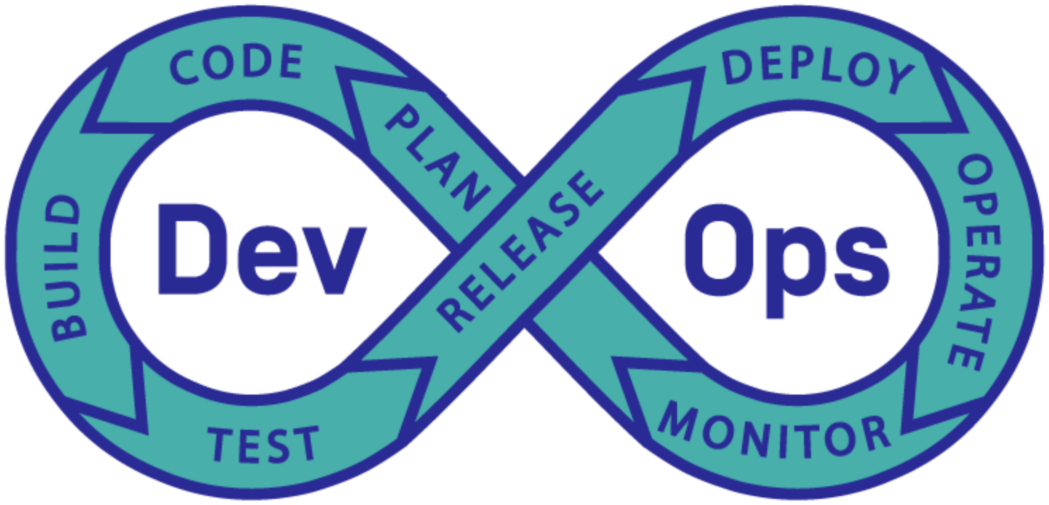
Using ICM lets you take advantage of the efficiency, agility, and repeatability provided by virtual and cloud computing and containerized software without major development or retooling. The InterSystems IRIS configurations ICM can provision and deploy range from a stand-alone instance, through load-balanced application servers connected to a data server in a distributed cache cluster, to a basic or complex sharded cluster. ICM can also deploy InterSystems IRIS on existing virtual and physical clusters.

Even if you are already using cloud infrastructure, containers, or both, ICM dramatically reduces the time and effort required to provision and deploy your application by automating numerous manual steps based on the information you provide. And the functionality of ICM is easily extended through the use of third-party tools and in-house scripting, increasing automation and further reducing effort.

Each element of the ICM approach provides its own advantages, which combine with each other:

- Configuration file templates allow you to accept default values provided by InterSystems for most settings, customizing only those required to meet your specific needs.
- The command line interface allows you to initiate each phase of the provisioning and deployment process with a single simple command, and to interact with deployed containers in a wide variety of ways.
- IaC brings the ability to quickly provision consistent, repeatable platforms that are easily reproduced, managed, and disposed of.
- IaaS providers enable you to utilize infrastructure in the most efficient manner — for example, if you need a cloud configuration for only a few hours, you pay for only a few hours — while also supporting repeatability, and providing all the resources you need to go with your compute nodes, such as networking and security, load balancers, and storage volumes.
- Containerized application deployment means seamlessly replaceable application environments on immutable software-provisioned infrastructure, separating code from data and avoiding the risks and costs of updating the infrastructure itself while supporting Continuous Integration/Continuous Deployment (CI/CD) and a DevOps approach.

Figure 1–2: Containers Support the DevOps Approach



You can work with InterSystems IRIS container images provided by InterSystems to deploy InterSystems IRIS, or as the base for your own application images.

ICM exploits these advantages to bring you the following benefits:

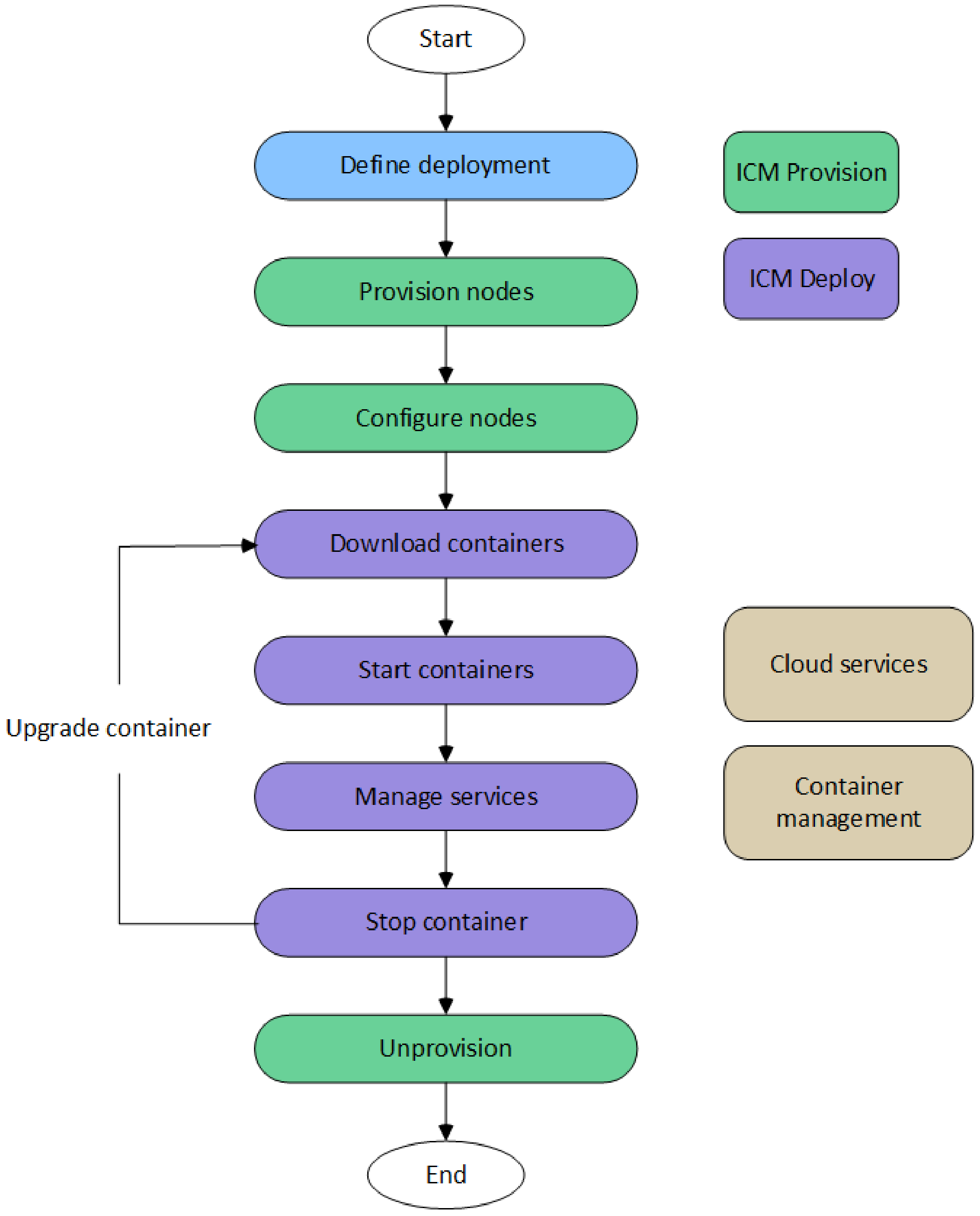
- Automated provisioning and deployment, and command-line management, of large-scale, cloud-based InterSystems IRIS configurations.
- Integration of existing InterSystems IRIS and InterSystems IRIS-based applications into your enterprise's DevOps toolchain.
- Increased agility through fast reprovisioning and redeployment when required.
- Stability, robustness, and minimization of risk through easy versioning of both the application and the environment it runs in.

Note: For a brief introduction to the use of InterSystems IRIS in Docker containers, including a hands-on experience, see *First Look: InterSystems IRIS in Docker Containers*; for detailed information about this topic, see *Running InterSystems IRIS in Containers*.

1.2 The ICM Application Lifecycle

The role of ICM in the application lifecycle, including its two main phases, *provision* and *deploy*, is shown in the following illustration:

Figure 1–3: Role of ICM in the Application Lifecycle



1.2.1 Define Goals

ICM's configuration files, as provided, contain almost all of the settings you need to provide to provision and deploy the InterSystems IRIS configuration you want. Simply define your desired configuration in the appropriate file, as well as specifying some details such as credentials (cloud server provider, SSH, SSL/TLS) , InterSystems IRIS licenses, types and sizes of the compute nodes you want, and so on. (See Define the Deployment for details.)

Note: In this document, the term *compute node* is used to refer to a virtual host provisioned either in the public cloud of one of the supported cloud service providers or in a private cloud using VMware vSphere.

1.2.2 Provision

ICM supports three main provisioning activities: the creation (provisioning), configuration, and destruction (unprovisioning) of compute nodes and associated resources in a cloud environment.

ICM carries out provisioning tasks by making calls to HashiCorp's Terraform. Terraform is an open source tool for building, changing, and versioning infrastructure safely and efficiently, and is compatible with both existing cloud services providers and custom solutions. Configuration files describe the provisioned infrastructure. (See Provision the Infrastructure for details.)

Although all of the tasks could be issued as individual Terraform commands, executing Terraform jobs through ICM has the following advantages over invoking Terraform directly:

Terraform Executed Directly	Terraform Executed by ICM
Configures each type of node in sequence, leading to long provisioning times	Runs multiple Terraform jobs in parallel to configure all node types simultaneously, for faster provisioning
Does not provide programmatic access (has no API)	Provides programmatic access to Terraform
Defines the desired infrastructure in the proprietary HashiCorp Configuration Language (HCL)	Defines the desired infrastructure in a generic JSON format

ICM also carries out some postprovisioning configuration tasks using SSH in the same fashion, running commands in parallel on multiple nodes for faster execution.

1.2.3 Deploy

ICM deploys prebuilt InterSystems IRIS images in Docker containers on the compute nodes it provisions. These containers are platform-independent and fully portable, do not need to be installed, and are easily tunable. ICM itself is deployed in a Docker container. A containerized application runs natively on the kernel of the host system, while the container provides it with only the elements needed to run it and make it accessible to the required connections, services, and interfaces — a runtime environment, the code, libraries, environment variables, and configuration files.

Deployment tasks are carried out by making calls to Docker. Although all of the tasks could be issued as individual Docker commands, executing Docker commands through ICM has the following advantages over invoking Docker directly:

- ICM runs Docker commands across all machines in parallel threads, reducing the total amount of time to carry out lengthy tasks, such as pulling (downloading) images.
- ICM can orchestrate tasks, such as rolling upgrades, that have application-specific requirements.

For detailed information about running InterSystems IRIS and InterSystems IRIS-based applications in Docker containers, see *Running InterSystems IRIS in Containers*.

1.2.4 Manage

ICM commands let you interact with and manage your infrastructure and containers in a number of ways. For example, you can run commands on the cloud hosts or within the containers, copy files to the hosts or the containers, upgrade containers, and interact directly with InterSystems IRIS.

For complete information about ICM service deployment and management, see Deploy and Manage Services.

2
Essential ICM Elements

The chapter describes the essential elements involved in using ICM, including the following:

- ICM Image
- Provisioning Platforms
- Deployment Platforms
- Node Types and InterSystems IRIS Architecture
- Field Values
- Command Line
- Configuration, State, and Log Files
- Docker Repositories

2.1 ICM Image

ICM is provided as a Docker image, which you run to start the ICM container. Everything required by ICM to carry out its provisioning, deployment, and management tasks — for example Terraform, the Docker client, and templates for the configuration files — is included in this container. See Launch ICM for more information about the ICM container.

The system on which you launch ICM must be supported by Docker as a platform for hosting Docker containers, have Docker installed, and have adequate connectivity to the cloud platform on which you intend to provision infrastructure and deploy containers.

2.2 Provisioning Platforms

ICM can provision virtual compute nodes and associated resources on the following platforms:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure)
- VMware vSphere (vSphere)

Note: To address the needs of the many users who rely on VMware vSphere, it is supported by this release of ICM. Depending on your particular vSphere configuration and underlying hardware platform, the use of ICM to provision virtual machines may entail additional extensions and adjustments not covered in this guide, especially for larger and more complex deployments, and may not be suitable for production use. Full support is expected in a later release.

Before using ICM with one of these platforms, you should be generally familiar with the platform. You will also need account credentials; see Obtain Security Credentials for more information.

ICM can also configure existing virtual or physical clusters (provider PreExisting) as needed and deploy containers on them, just as with the nodes it provisions itself.

2.3 Deployment Platforms

ICM supports deployment of containers on nodes running enterprise-level operating system platforms supported for the purpose by InterSystems, currently Red Hat Enterprise Linux, version 7.4 or 7.5, Ubuntu 18.04 or later, and SUSE Enterprise Linux 12.3 or later.

Docker images created by InterSystems are supported on the Supported Server Platforms listed in *InterSystems Support Platforms* and on Red Hat Enterprise Linux, SUSE Enterprise Linux, Ubuntu, and CentOS for development.

2.4 Node Types and InterSystems IRIS Architecture

ICM deploys one InterSystems IRIS™ instance per provisioned node, and the role that each instance plays in an InterSystems IRIS configuration is determined by the Role field value under which the node and the instance on it are provisioned, deployed, and configured.

2.4.1 Defining Nodes in the Deployment

In preparing to use ICM, you must define your target configuration (see Define the Deployment) by selecting the types and numbers of nodes you want to include. If you want to deploy an InterSystems IRIS sharded cluster, for example, you must decide beforehand how many shard data servers (role DS), shard query servers (role QS), if any, and shard master application servers (role AM), if any, will be included in the cluster, and whether the shard master data server (role DM) and shard data servers are to be mirrored. The specifications for the desired nodes are then entered in the definitions file (see Configuration, State, and Log Files) to provide the needed input to ICM. This is shown in the following simplified example defining a shard master data server, six shard data servers, and three shard master application servers:

```
[
    {
        "Role": "DM",
        "Count": "1"
    },
    {
        "Role": "DS",
        "Count": "6"
    },
    {
        "Role": "AM",
        "Count": "3"
}
]
```

The Mirror field, which determines whether mirroring is configured, appears in the defaults file; it is covered at length in ICM Cluster Topology.

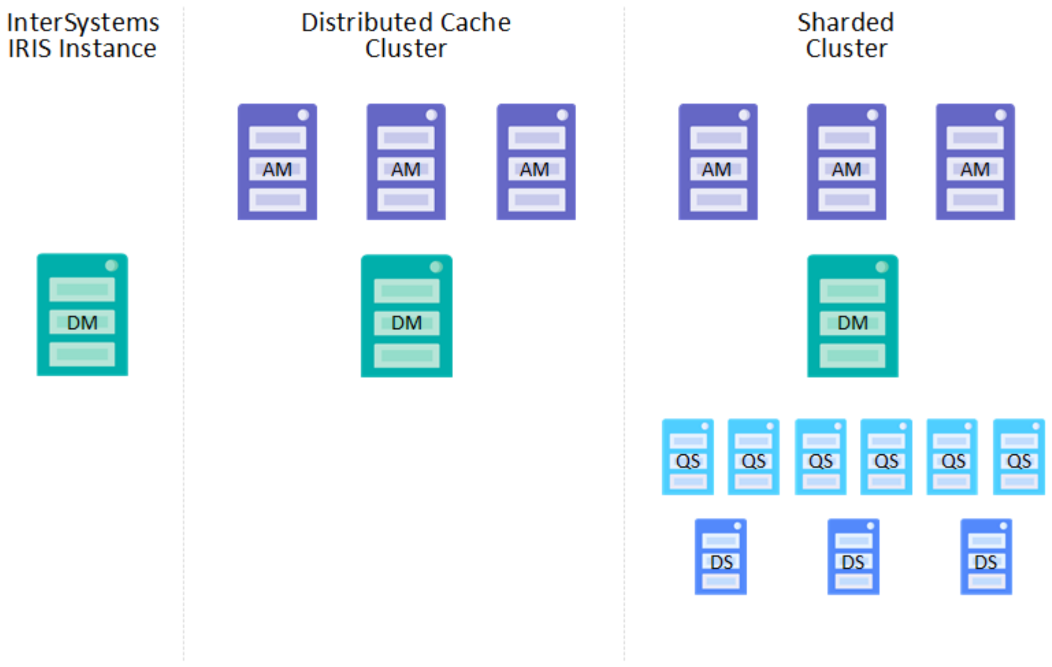
2.4.2 Defining Sharded or Traditional InterSystems IRIS Architecture

ICM gives you the choice of deploying an InterSystems IRIS sharded cluster, an InterSystems IRIS distributed cache cluster, or a stand-alone InterSystems IRIS instance. This is possible because two node types, DM and AM, can represent different roles in the configuration depending on what is deployed with them, as follows:

- In a sharded cluster, the DM node represents the shard master data server and AM nodes represent shard master application servers, along with the shard data servers (DS nodes) and optionally the shard query servers (QS nodes).
- If a DM node and AM nodes are deployed without shard data servers (DS nodes), the DM node is instead configured as the data server in a distributed cache cluster, and the AM nodes as application servers.
- If a DM node is deployed without either AM or DS nodes, it becomes a stand-alone InterSystems IRIS instance.

These configurations are shown in the following illustration:

Figure 2–1: Stand-alone, Distributed Cache, and Sharded Configurations as Deployed by ICM



ICM also provisions and deploys web servers and load balancers as needed, and arbiter nodes as part of mirrored configurations.

For more information about ICM node types and the ways in which they can be configured together, see ICM Node Types and ICM Cluster Topology. For detailed information about InterSystems IRIS sharded and distributed caching architectures, see the *Scalability Guide*.

2.5 Field Values

The provisioning, deployment, and configuration work done by ICM is determined by a number of field values that you provide. For example, the Provider field specifies the provisioning platform to use; if its value is AWS, the specified nodes are provisioned on AWS.

The are two ways to specify field values:

- Some can be specified on the command line (see Command Line).
- All can be included in the two configuration files (see Configuration, State and Log Files), although some can appear only in the defaults.json file, and others intended for the definitions file only..

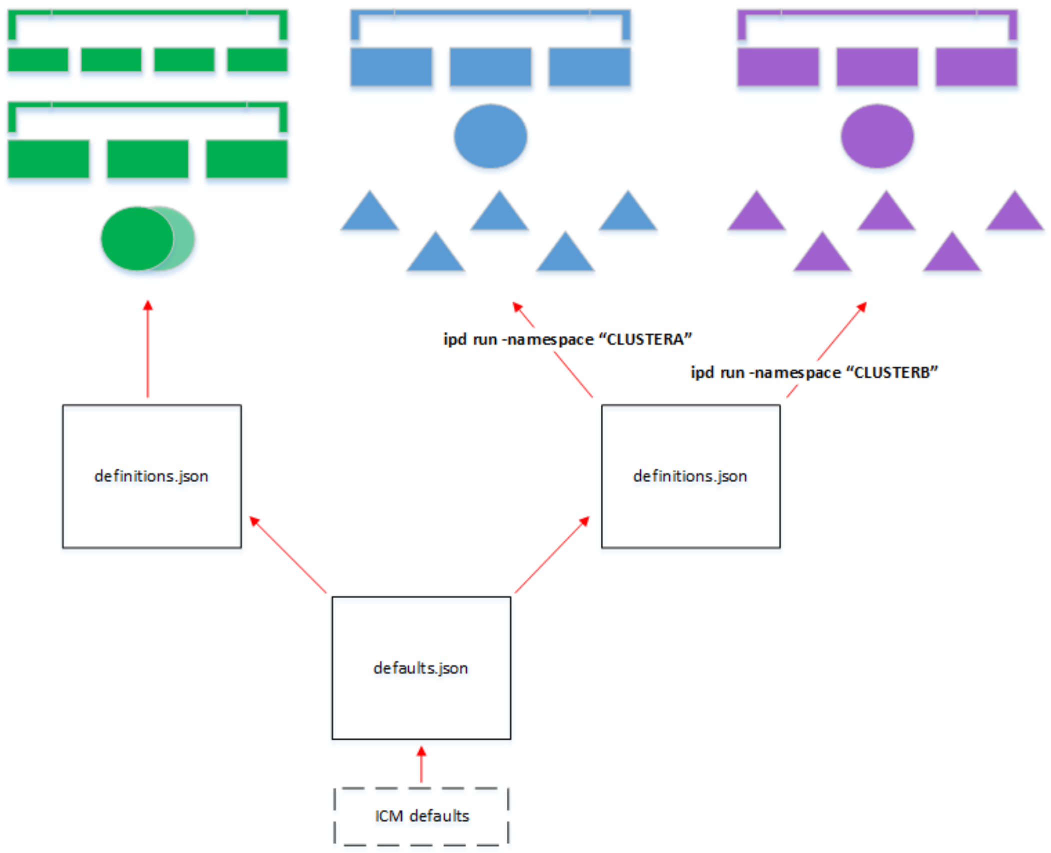
There are also defaults for most ICM fields. In descending order of precedence, these specifications are ranked as follows:

1. Command line option
2. Entry in definitions.json configuration file
3. Entry in defaults.json configuration file
4. ICM default value

This avoids repeating commonly used fields while allowing flexibility. The defaults file (defaults.json) can be used to provide values (when a value is required or to override the defaults) for multiple deployments in a particular category, for example those that are provisioned on the same platform. The definitions file definitions.json) provides values for a particular deployment, such as specifying the nodes in the cluster and the labels that identify them, but can also contain fields included in the defaults file, in order to override the defaults.json value for a particular node type in a particular deployment. Specifying a field value as a command line options let you override both configuration files in the context of a particular command in a particular deployment.

The following illustration shows three deployments sharing a single defaults.json file, while two also share a definitions.json file. For the latter, different InterSystems IRIS namespaces are created during deployment by adding an option to the **icm run** command.

Figure 2–2: ICM Configuration Files Define Deployments



For a list of all fields and definitions of their values, see ICM Reference.

2.6 Command Line

The ICM command line interface allows users to specify an action in the orchestration process — for example, **icm provision** to provision infrastructure, or **icm run** to deploy by running the specified containers — or a container or service management command, such as upgrading a container or connecting to InterSystems IRIS. ICM executes these commands using information from configuration files. If either of the input configuration files (see Configuration, State and Log Files) is not specified on the command line, ICM uses the file (definitions.json or defaults.json) that exists in the current working directory; if one or both are not specified and do not exist, the command fails. For a complete list of ICM commands, see ICM Reference.

The command line can be also used to specify options, which have several purposes, as follows:

- To override information that is in a configuration file for a single command only, for example to deploy a different Docker image than the one provided in the configuration files, as shown in the following:

  ```
  icm run –image image
  ```

- To supply information that is not in the configuration files because you do not want it persisted, for instance a password, as follows:

  ```
  icm run -iscPassword password
  ```

- To supply information relevant to a command executed on one or more nodes in the deployment, for example a query, as shown:

```
icm sql -role DM -namespace namespace -command "query"
```

Options can have different purposes when used with different commands. For example, with the **icm run** command, the **-container** option provides the name for the new container being started from the specified image, whereas with the **icm ps** command it specifies the name of the existing deployed containers for which you want to display run state information.

For comprehensive lists of ICM commands and command-line options, see ICM Commands and Options.

2.7 Configuration, State, and Log Files

ICM uses JSON files as both input and output, as follows:

- As input, configuration files provide the information ICM needs to execute tasks, such as the types and numbers of nodes to provision. These files are provided by the user; they can be adapted from the template provided with ICM, created manually, or produced by the output of a script or UI.
- As output, files generated by ICM describe the results of ICM's tasks.

When an ICM task results in the generation of new or changed data (for example, an IP address), that information is recorded in JSON format for use by subsequent tasks. ICM ignores all fields which it does not recognize or that do not apply to the current task, but passes these fields on to subsequent tasks, rather than generating an error. This behavior has the following advantages:

- Information specified during an earlier phase (such as provisioning) to be used during a later phase (such as deployment) without having to edit or maintain more than one configuration file
- A degree of forward- and backward-compatibility among ICM versions is supported.
- The work required to use one configuration file with multiple providers is minimized.
- Although the JSON standard does not provide a formal means of commenting out content, fields can be "commented out" by altering their names.

The JSON files used by ICM include the following:

- The definitions file and the defaults file, provided by the user as input to ICM's provisioning and deployment phases. By default these files are assumed to be in the current working directory.
- The instances file, generated by ICM at the end of the provisioning phase and used as input to the deployment phase. By default this file is created in the current working directory.
- The Terraform state files, generated by ICM during the provisioning phase and used as input to future infrastructure-related operations, such as unprovisioning. By default these files are placed in a state directory generated by ICM under the current working directory.

ICM also generates a number of other log, output, and error files, which are located in the current working directory or the state directory.

2.7.1 The Definitions File

The definitions file (definitions.json) describes a set of compute nodes to be provisioned for a particular deployment. The file consists of a series of JSON objects representing node definitions, each of which contains a list of attributes as well as a count to indicate how many nodes of that type should be created. Some fields are required, others are optional, and some are provider-specific (that is, for use with AWS, Google Cloud Platform, Microsoft Azure or VMware vSphere).

Most fields can appear in this file, repeated as need for each node definition. Some fields, however, must be the same across a single deployed configuration, and therefore cannot be changed from the default, or specified if there is no default, by entries in the definitions file. The Provider field is a good example, for obvious reasons. Another field that must be in the defaults file is Namespace, which specifies the user namespace to create on deployed InterSystems IRIS instances; default globals and routines databases of the same name are created as well. (If unspecified, Namespace defaults to USER.) The user namespace also becomes the default namespace for the **icm session** and **icm sql** commands, and its default globals database is used for persistent storage (outside the deployed container) of instance-specific InterSystems IRIS data (see Durable %SYS for Persistent Instance Data in *Running InterSystems IRIS in Containers*). Other fields that cannot be included in the definitions.json file and will cause an error if they are include Label and Tag.

The following shows the contents of a sample definitions.json file for provisioning a distributed cache cluster consisting of a data server (**"Role": "DM"**"), three application servers (**"Role": "AM"**), and a mirror arbiter node (**"Role": "AR"**) on AWS.

Fields that vary between node types (for example, Role) must be included in the node definitions in definitions.json. The definitions.json is sometimes used to override either ICM defaults or settings in the defaults.json. For instance, in the following example:

- The DataVolumeSize field appears here only for the DM nodes here because the others use the ICM default value.
- The DM node and AR node definitions include an InstanceType field overriding the default instance type specified in defaults.json.
- The AR node definition includes a DockerImage field overriding the one in defaults.json because it an arbiter container will be deployed on it, rather than an InterSystems IRIS container.

Some fields must be in definitions.json because they are restricted to certain node types; for example, if LoadBalancer is set to True for nodes of type application server (AM) or web server (WS), a load balancer is automatically provisioned, but applying this setting to other node types causes errors.

```
[
    {
        "Role": "DM",
        "Count": "2",
        "DataVolumeSize": "50",
        "InstanceType": "m4.xlarge"
    },
    {
        "Role": "AM",
        "Count": "3",
        "StartCount": "3",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "StartCount": "4",
        "InstanceType": "t2.small",
        "DockerImage": "intersystems/arbiter:stable"
    }
]
```

2.7.2 The Defaults File

Generally, the defaults file defines fields that are the same across all deployments of a particular type, such as those provisioned on a particular cloud platform.

As noted in The Definitions File, while most fields can be in either input file, some must be the same across a deployment and cannot be specified separately for each node type, for example Provider. In addition to these, there may be other fields that you want to apply to all nodes in all deployments, overriding them when desired on the command line or in the definitions file. Fields of both types are included in the defaults.json file. Including as many fields as you can in the defaults file keeps definitions files smaller and more manageable.

The format of the defaults file is a single JSON object; the values it contains are applied to every field whose value is not specified (or is null) in a command line option or the definitions file.

The following shows the contents of a sample defaults.json file used with the sample definitions.json file provided in The Definitions File. Some of the defaults specified in the former are overridden by the latter, including OSVolumeSize, DataVolumeSize, and InstanceType.

```
{
    "Provider": "AWS",
    "Label": "ACME",
    "Tag": "TEST",
    "DataVolumeSize": "10",
    "SSHUser": "ec2-user",
    "SSHPublicKey": "/Samples/ssh/insecure-ssh2.pub",
    "SSHPrivateKey": "/Samples/ssh/insecure",
    "DockerImage": "intersystems/iris:stable",
    "DockerUsername": "xxxxxxxxxxxx",
    "DockerPassword": "xxxxxxxxxxxx",
    "DockerVersion": "ce-18.03.1.ce",
    "TLSKeyDir": "/Samples/tls/",
    "LicenseDir": "/Samples/license/",
    "Region": "us-west-1",
    "Zone": "us-west-1c",
    "AMI": "ami-18726478",
    "InstanceType": "m4.large",
    "Credentials": "/Samples/AWS/sample.credentials",
    "SystemMode": "TEST",
    "ISCPassword": "",
    "Namespace": "DB",
    "Mirror": "false"}
```

2.7.3 The Instances File

The instances file (instances.json), generated by ICM during the provisioning phase, describes the set of compute nodes that have been successfully provisioned. This information provides input to the deployment and management phase, and the file must therefore be available during this phase, and its path provided to ICM if it is not in the current working directory. By default, the instances file is created in the current working directory; you can change this using the **-instances** option, but note that if you do you must supply the alternate location by using the **-instances** option with all subsequent commands.

While the definitions file contains only one entry for each node type, including a Count value to specify the number of nodes of that type, the instances file contains an entry for reach node actually provisioned. For example, the sample definitions file provided earlier contains three entries — one for three application servers, one for two data servers, and one for an arbiter — but the resulting instances file would contain six objects, one for each provisioned node.

All of the parameters making up each node's definition — including those in the definitions and defaults file, those not specified in the configuration files that have default values, and internal ICM parameters — appear in its entry, along with the node's machine name constructed from the Label, Role, and Tag fields), its IP address, and its DNS name. The location of the subdirectory for that node in the deployment's state directory is also included.

2.7.4 The State Directory and State Files

ICM writes several state files, including logs and generated scripts, to a unique subdirectory for use during the lifetime of the provisioned infrastructure. By default, this subdirectory is created in the current working directory, with a name of the form ICM-*UID*, for example:

```
./ICM-172807747058302123/
```

State files generated during provisioning include the Terraform overrides file and state file, terraform.tfvars and terraform.tfstate, as well as Terraform output, error and log files. A set of these Terraform-related files is created in a separate subdirectory for each node type definition in the definitions file, for example:

```
./ICM-172807747058302123/ACME-DM-TEST
./ICM-172807747058302123/ACME-AM-TEST
./ICM-172807747058302123/ACME-AR-TEST
```

Important: ICM relies on the state files it creates for accurate, up to date information about the infrastructure it has provisione; without them, the provisioning state may be difficult or impossible to for ICM to reconstruct, resulting in errors, and perhaps even the need for manual intervention. For this reason, InterSystems strongly recommends making sure the state directory is located on storage that is reliable and reliably accessible, with an appropriate backup mechanism in place. Note that if you use the **-stateDir** command line option with the **icm provision** command to override the default location, you must continue using the **-stateDir** option to specify that location in all subsequent provisioning commands.

2.7.5 Log Files and Other ICM Files

ICM writes several log, output and error files to the current working directory and within the state directory tree. The icm.log file in the current working directory records ICM's informational, warning, and error messages. Other files within the state directory tree record the results of commands, including errors. For example, errors during the provisioning phase are typically recorded in the terraform.err file.

Important: When an error occurs during an ICM operation, ICM displays a message directing you to the log file in which information about the error can be found. Before beginning an ICM deployment, familiarize yourself with the log files and their locations.

2.8 Docker Repositories

Each image deployed by ICM is pulled (downloaded) from a Docker repository. Many Docker images can be freely downloaded from public Docker repositories; private repositories such as the InterSystems repository, however, require a Docker login.

2.8.1 Logging Into a Docker Repository

As part of the deployment phase, ICM logs each node into the Docker respository you specify, using credentials supplied by you, before deploying the image specified by the DockerImage field in one of the configuration files or on the command line using the **-image option**. (The repository name must be included in the image specification.) You can include the following three fields in the defaults.json file to provide the needed information:

- DockerRegistry

  The DNS name of the server hosting the Docker repository storing the image specified by DockerImage. If this field is not included, ICM uses Docker's public registry located at registry-1.docker.io.

  If the repository specified by DockerImage does not exist on the server specified by DockerRegistry, deployment fails and returns an error.

- DockerUsername

  The username to use for Docker login. Not required for public repositories. If this field is not included and the repository specified by DockerImage is private, login fails.

- DockerPassword

  The password to use for Docker login. Not required for public repositories. If this field is not included and the repository specified by DockerImage is private. ICM prompts you (with masked input) for a password.

  Note: If the value of the DockerPassword field contains special characters such as **$**, **|**, **(**, and **)**, they must be escaped with two **** characters; for example, the password **abc$def** must be specified as **abc\\\\$def**.

2.8.2 Setting Up a Docker Repository

You may want to set up a Docker repository so you can store InterSystems images (and your own images) locally rather than relying on the network availability for critical applications. For information on doing this, see Deploy a registry server in the Docker documentation.

3
Using ICM

This chapter explains how to use ICM to deploy an InterSystems IRIS™ configuration in a public cloud, as follows. The sections that follow explain the steps involved in using ICM to deploy a sample InterSystems IRIS configuration on AWS, as follows:

- ICM Use Cases

  Review deployments based on two sample use cases that are used as examples throughout the chapter.

- Launch ICM

  Use the **docker run** command with the ICM image provided by InterSystems to start the ICM container and open a command line.

- Obtain Security Files

  Obtain the cloud provider and SSL/TLS credentials needed for secure communications by ICM.

- Define the Deployment

  Decide how many of each nodes type you want to provision and make other needed decisions, then create the defaults.json and definitions.json configuration files needed for the deployment (you can use the provided templates for this purpose).

- Provision the Infrastructure

  Use the **icm provision** command to provision the infrastructure and explore ICM's infrastructure management commands.

- Deploy and Manage Services

  Run the **icm run** command to deploy containers, and explore ICM's container and service management commands.

- Unprovision the Infrastructure

  Run the **icm unprovision** command to destroy the deployment.

For comprehensive lists of the ICM commands and command-line options covered in detail in the following sections, see ICM Commands and Options.

3.1 ICM Use Cases

This chapter is focused on two typical ICM use cases, deploying the following two InterSystems IRIS configurations:

- Distributed cache cluster — Mirrored data server, three application servers, arbiter node, and load balancer. This deployment is illustrated in the section Distributed Cache Cluster Definitions File.

- Basic sharded cluster — Shard master data server and three shard data servers. This deployment is illustrated in the section Sharded Cluster Definitions File.

Most of the steps in the deployment process are the same for both configurations. The primary difference lies in the definitions files; see Define the Deployment for detailed contents. Output shown for the provisioning phase (see The icm provision Command) is from the distributed cache cluster; output shown for the deployment phase (see Deploy and Manage Services) is for the sharded cluster.

3.2 Launch ICM

ICM is provided as a Docker image. Everything required by ICM to carry out its provisioning, deployment, and management tasks — for example Terraform, the Docker client, and templates for the configuration files — is included in the ICM container. Therefore the only requirement for the Linux, macOS or Microsoft Windows system on which you launch ICM is that Docker is installed.

- Identifying the Repository and Image
- Running the ICM Container

Important: ICM is supported on Docker Enterprise Edition and Community Edition version 18.09 and later; Enterprise Edition only is supported for production environments.

Not all combinations of platform and Docker version are supported by Docker; for detailed information from Docker on compatibility, see the Compatibility Matrix and About Docker CE.

3.2.1 Identifying the Repository and Image

To download and run the ICM image, and to deploy the InterSystems IRIS container and others in the cloud using ICM, you need to identify the repository in which these InterSystems images are located and the credentials you need to log into that repository. The repository must be accessible from the internet (that is, not behind a firewall) in order for the cloud provider to download images.

InterSystems images are distributed as Docker tar archive files, available in the InterSystems Worldwide Response Center (WRC) download area. Your enterprise may have already have added these images to its Docker repository; in this case, you should get the location of the repository and the needed credentials from the appropriate IT administrator. If your enterprise has a Docker repository but has not yet added the InterSystems images, get the location of the repository and the needed credentials, obtain the tar archive files containing the ICM and InterSystems IRIS images from the WRC, and add each of them to the repository using the following steps on the command line:

1. **docker load -i** *archive_file*
2. **docker tag docker.intersystems.com/intersystems/***image_name***:***version your_repository/image_name***:***version*

   For example:

   ```
   docker tag docker.intersystems.com/intersystems/icm:2018.1.0.583 acme/icm:2018.1.0.583
   ```

3. **docker login** *your_repository*

   For example:

   ```
   docker login docker.acme.com
   Username: gsanchez@acme.com
   Pasword: **********
   ```

4. **docker push** *your_repository/image_name***:***version*

For example:

```
docker push acme/icm:2018.1.0.583
```

If your organization does not have an internet accessible Docker repository, you can use the free (or extremely low cost) Docker Hub for your testing.

3.2.2 Running the ICM Container

To launch ICM from the command line on a system on which Docker is installed, use the **docker run** command (which actually combines three separate Docker commands) to do the following:

- Download the ICM image from the repository if it is not already present locally (can be done separately with the **docker pull** command).
- Creates a container from the ICM image (**docker create** command).
- Start the ICM container (**docker start** command).

For example:

```
docker run --name icm -it --cap-add SYS_TIME intersystems/icm:2018.1.0.583
```

The **-i** option makes the command interactive and the **-t** option opens a pseudo-TTY, giving you command line access to the container. From this point on, you can interact with ICM by invoking ICM commands on the pseudo-TTY command line. The **--cap-add SYS_TIME** option allows the container to interact with the clock on the host system, avoiding clock skew that may cause the cloud service provider to reject API commands.

The ICM container includes a /Samples directory that provides you with samples of the elements required by ICM for provisioning, configuration, and deployment. The /Samples directory makes it easy for you to provision and deploy using ICM out of the box. Eventually, you can use locations outside the container to store these elements and InterSystems IRIS licenses, and either mount those locations as external volumes when you launch ICM or copy files into the CIM container using the **docker cp** command.

Of course, the ICM image can also be run by custom tools and scripts, and this can help you accomplish goals such as making these external locations available within the container, and saving your configuration files and your state directory (which is required to remove the infrastructure and services you provision) to persistent storage the container as well. A script, for example, could do the latter by capturing the current working directory in a variable and using it to mount that directory as a storage volume when running the ICM container, as follows:

```
#!/bin/bash
clear

# extract the basename of the full pwd path
MOUNT=$(basename $(pwd))
docker run --name icm -it --volume $PWD:$MOUNT --cap-add SYS_TIME intersystems/icm:stable
printf "\nExited icm container\n"
printf "\nRemoving icm container...\nContainer removed:  "
docker rm icm
```

You can mount multiple external storage volumes when running the ICM container (or any other). For detailed information about the InterSystems IRIS feature that lets you store instance-specific data outside the container, see Durable %SYS for Persistent Instance Data in *Running InterSystems IRIS in Containers*.

Note: On a Windows host, you must enable the local drive on which the directory you want to mount as a volume is located using the **Shared Drives** option on the Docker **Settings ...** menu; see Using InterSystems IRIS Containers with Docker for Windows on InterSystems Developer Community for more information.

Important: When an error occurs during an ICM operation, ICM displays a message directing you to the log file in which information about the error can be found. Before beginning an ICM deployment, familiarize yourself with the log files and their locations as described in Log Files and Other ICM Files.

3.3 Obtain Security-Related Files

ICM communicates securely with the cloud provider on which it provisions the infrastructure, with the operating system of each provisioned node, and with Docker and several InterSystems IRIS services following container deployment. Before defining your deployment, you must obtain the credentials and other files needed to enable secure communication.

3.3.1 Cloud Provider Credentials

To use ICM with one of the public cloud platforms, you must create an account and download administrative credentials. To do this, follow the instructions provided by the cloud provider; you can also find information about how to download your credentials once your account exists in the Provider-Specific Parameters section. In the ICM configuration files, you identify the location of these credentials using the Credentials parameter.

When using ICM with a vSphere private cloud, you can use an existing account with the needed privileges, or create a new one. You specify these using the Username and Password fields.

3.3.2 SSH and SSL/TLS Keys

ICM uses SSH to provide secure access to the operating system of provisioned nodes, and SSL/TLS to establish secure connections to Docker, InterSystems Web Gateway, JDBC, and mirrored InterSystems IRIS databases. The locations of the files needed to enable this secure communication are specified using several ICM parameters, including:

- SSHPublicKey

  Public key of SSH public/private key pair used to enable secure connections to provisioned compute nodes; in SSH2 format for AWS and OpenSSH format for other providers.

- SSHPrivateKey

  Private key of SSH public/private key pair.

- TLSKeyDir

  Directory containing TLS keys used to establish secure connections to Docker, InterSystems Web Gateway, JDBC, and mirrored InterSystems IRIS databases.

You can create these files, either for use with ICM, or to review them in order to understand which are needed, using two scripts provided with ICM, located in the directory /ICM/bin in the ICM container. The keygenSSH.sh script creates the needed SSH files and places them in the directory /Samples/ssh in the ICM container. The keygenTLS.sh script creates the needed SSL/TLS files and places them in /Samples/tls. You can then specify these locations when defining your deployment, or obtain your own files based on the contents of these directories.

For more information about the security files required by ICM and generated by the keygen* scripts, see ICM Security and Security-Related Parameters in the “ICM Reference” chapter.

Important: The keys generated by these scripts, as well as your cloud provider credentials, must be fully secured, as they provide full access to any ICM deployments in which they are used.

The keys by the keygen* scripts are intended as a convenience for your use in your initial test deployments. (Some have strings specific to InterSystems Corporation.) In production, the needed keys should be generated or obtained in keeping with your company's security policies.

3.4 Define the Deployment

To provide the needed parameters to ICM, you must select values for a number of fields, based on your goal and circumstances, and then incorporate these into the defaults and definitions files to be used for your deployment. (See Configuration, State and Log Files for information about these files.) You can begin with the template defaults.json and definitions.json files provided in the ICM container in the /Samples directory tree, for example /Samples/AWS.

The difference in the deployment processes for the two target configurations described at the start of this chapter lies primarily in their separate definitions files. Field values they can share are included in a joint defaults file, while the nodes that must be provisioned and configured for each deployment are specified in their definitions files.

The following sections provide the content of both the shared defaults file and the separate definitions files. Each field/value pair is shown as it would appear in the configuration file. Note that the fields included here do not represent an exhaustive list of all applicable fields; see ICM Configuration Parameters for information about these and all ICM fields.

- Shared Defaults File
- Distributed Cache Cluster Definitions File
- Sharded Cluster Definitions File

In general, which fields are included in defaults.json and which in definitions.json depends on your needs, but as noted in Configuration, State and Log Files, defaults.json is often used to provide values for multiple deployments in a particular category, for example those that are provisioned on the same platform, while definitions.json provides values for a particular deployment.

In this case, because both deployments are on AWS, if we assume that they also use the same credentials, deploy the same InterSystems IRIS image, and include Weave Scope monitoring, those fields can be included in defaults.json, while the fields that differ go in definitions.json. The tables are ordered to reflect this approach. As previously noted, some fields, such as *Provider*, must appear in the defaults files. The separate definitions files can be specified in an option on the provisioning command line, or the appropriate file can be swapped in as definitions.json in the current working directory before the **icm provision** command is executed (see Provision the Infrastructure).

For more information about the parameters listed in these tables, see ICM Configuration Parameters.

Note: InterSystems IRIS is already installed in the InterSystems container (or the container you created based on it) deployed by ICM, and ICM automatically configures InterSystems IRIS after it is deployed, as needed for the role of each instance. There are, however, a number of ICM fields you can use to configure the InterSystems IRIS instances differently from the InterSystems IRIS and ICM defaults, for example the ISCglobals field used to configure the database caches of the instances defined in the sample Sharded Cluster Definitions File. Information about InterSystems IRIS configuration settings, their effects, and their installed defaults is provided in the Installation Guide, the System Administration Guide, and the InterSystems Parameter File Reference.

3.4.1 Shared Defaults File

The field/value pairs shown in the following table represent the contents of a defaults.json file that can be used for both the distributed cache cluster deployment and the sharded cluster deployment.

Note: The pathnames provided in the fields specifying security files in this sample defaults file assume you have placed your AWS credentials in the /Samples/AWS directory, and used the keygen*.sh scripts to generate the keys, as described in Obtain Security-Related Files. If you have generated or obtained your own keys, these may be internal paths to external storage volumes mounted when the ICM container is run, as described in the Launch ICM. See ICM Security and Security-Related Parameters for additional information about these files.

Requirement	Definition	Field: Value
Provisioning platform Platform details	Amazon Web Services, required details; see About AWS Regions and Availability Zones. .	"Provider": "AWS" "Zone": "us-west-1c" "Region": "us-west-1"
Default machine image and sizing	Default AWS AMI and instance type to provision. Default size of data storage volume (overriding ICM default). See .About AWS AMIs and About AWS instance types.	"AMI": "ami-18726478" "InstanceType": "m4.large" "DataVolumeSize": "20"
SSHUser	Nonroot account with **sudo** access used by ICM for access to provisioned nodes.	"SSHUser": "ec2-user" (For AWS, the required value depends on the AMI; see Provider-Specific Parameters for more information.)
Locations of security files	Needed credentials and key files; because provider is AWS, the SSH2–format public key in /Samples/ssh/ is specified.	"Credentials": "/Samples/AWS/credentials" "SSHPublicKey": "/Samples/ssh/secure-ssh2.pub" "SSHPrivateKey": "/Samples/ssh/secure-ssh2" "TLSKeyDir": "/Samples/tls/"
Location of InterSystems IRIS license keys	License keys to be served to the InterSystems IRIS instances deployed on the provisioned nodes.	“LicenseDir”: “/Samples/Licenses”
Monitoring	Install Weave Scope, optionally provide authentication credentials; see Monitoring in ICM in the “ICM Reference” chapter.	"Monitor": "scope" “ProxyImage”: "intersystems/https-proxy-auth:stable" “MonitorUsername”: "..." “MonitorPassword”: "..."
Image to deploy, repository credentials, version of Docker to install on nodes	Latest InterSystems IRIS image, credentials to log into Docker repository, version of Docker to install; see Identifying the Repository and Image and General Parameters.	"DockerImage": "intersystems/iris:stable" "DockerUsername": "..." "DockerPassword": "..." "DockerVersion": "ce-18.09.1.ce"
Naming scheme for provisioned nodes	ACME-role-TEST-*NNNN*	"Label": "ACME" "Tag": "TEST"
InterSystems IRIS settings	Password for deployed instances.	Password is specified on the deployment command line (see Deploy and Manage Services) to avoid displaying the password in a configuration file

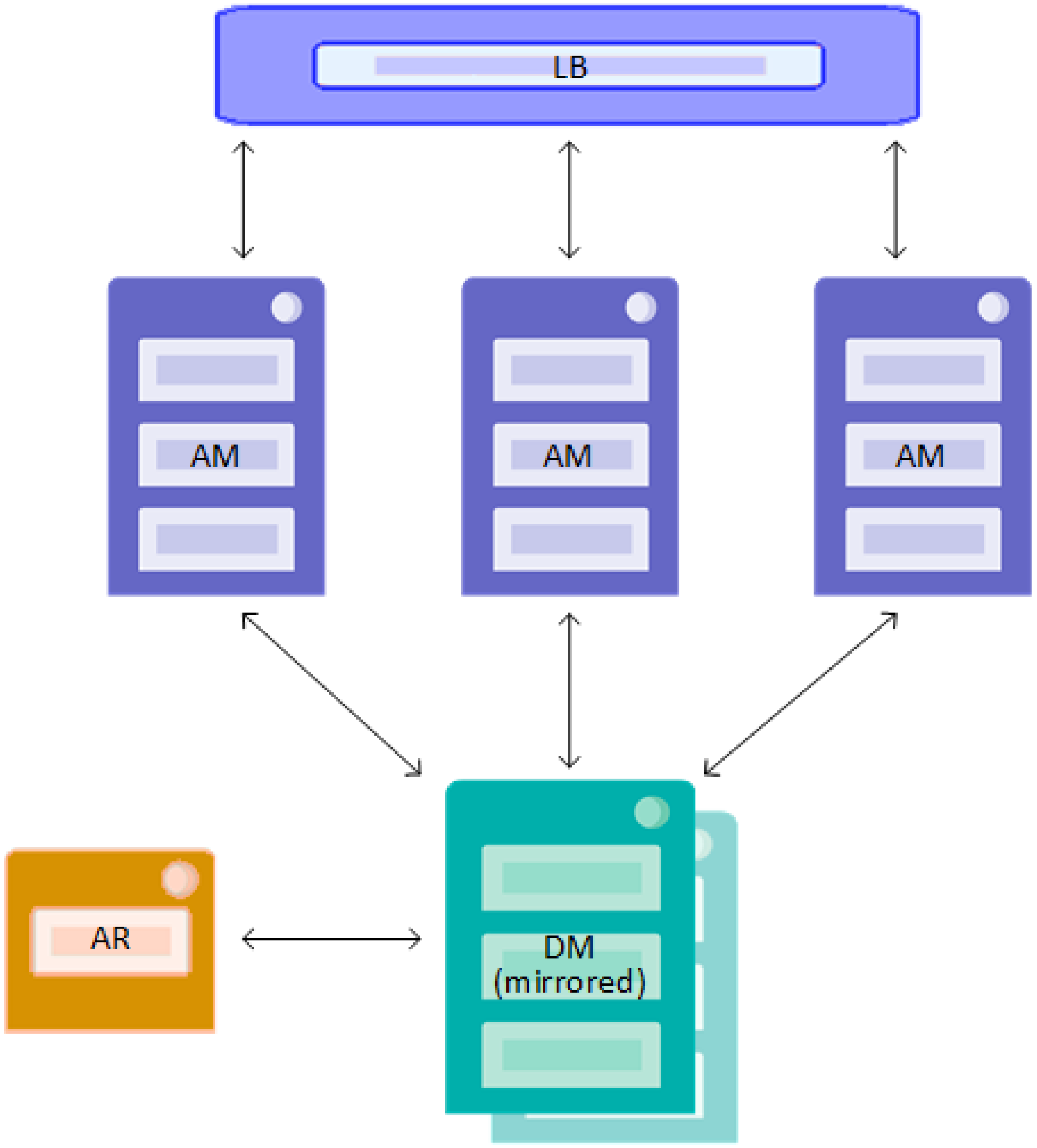
3.4.2 Distributed Cache Cluster Definitions File

The definitions.json file for the distributed cache cluster must define the following nodes:

- Two data servers (role DM), configured as a mirror
- Three application servers (role AM)
- Load balancer for application servers
- Arbiter node for data server mirror

This configuration is illustrated in the following:

Figure 3–1: Distributed Cache Cluster to be Deployed by ICM



In addition, the Mirror field must be set to True in the definitions file, as it is not used in the sharded cluster deployment and thus cannot appear in the defaults file.

The table that follows lists the field/value pairs that are required for this configuration.

Requirement	Definition	Field: Value
Mirroring	If Mirror is True, when two DM nodes are defined, they are mirrored.	"Mirror": "true"
Nodes for target InterSystems IRIS configuration, including provider-specific characteristics	Two data servers using a standard InterSystems IRIS license. Instance type and data volume size override defaults file (see Shared Defaults File);. Automatically configured as a mirror due to Mirror setting (previous row).	"Role": "DM" "Count": "2" "LicenseKey": "standard-iris.key" "InstanceType": "m4.xlarge" "OSVolumeSize": "32" "DataVolumeSize": "15"
	Three load-balanced application servers using a standard InterSystems IRIS license, naming starts at 0003. Load balancer is provisioned automatically when LoadBalancer is True.	"Role": "AM" "Count": "3" "StartCount": "3" "LicenseKey": "standard-iris.key" "LoadBalancer": "true"
	Arbiter node for data server mirror. Use of **arbiter** Docker image overrides **iris** image specified in defaults file. No license needed, naming is 0006. Instance type overrides defaults file.	"Role": "AR" "Count": "1" "StartCount": "6" "DockerImage": "intersystems/arbiter:stable" "InstanceType": "t2.small"

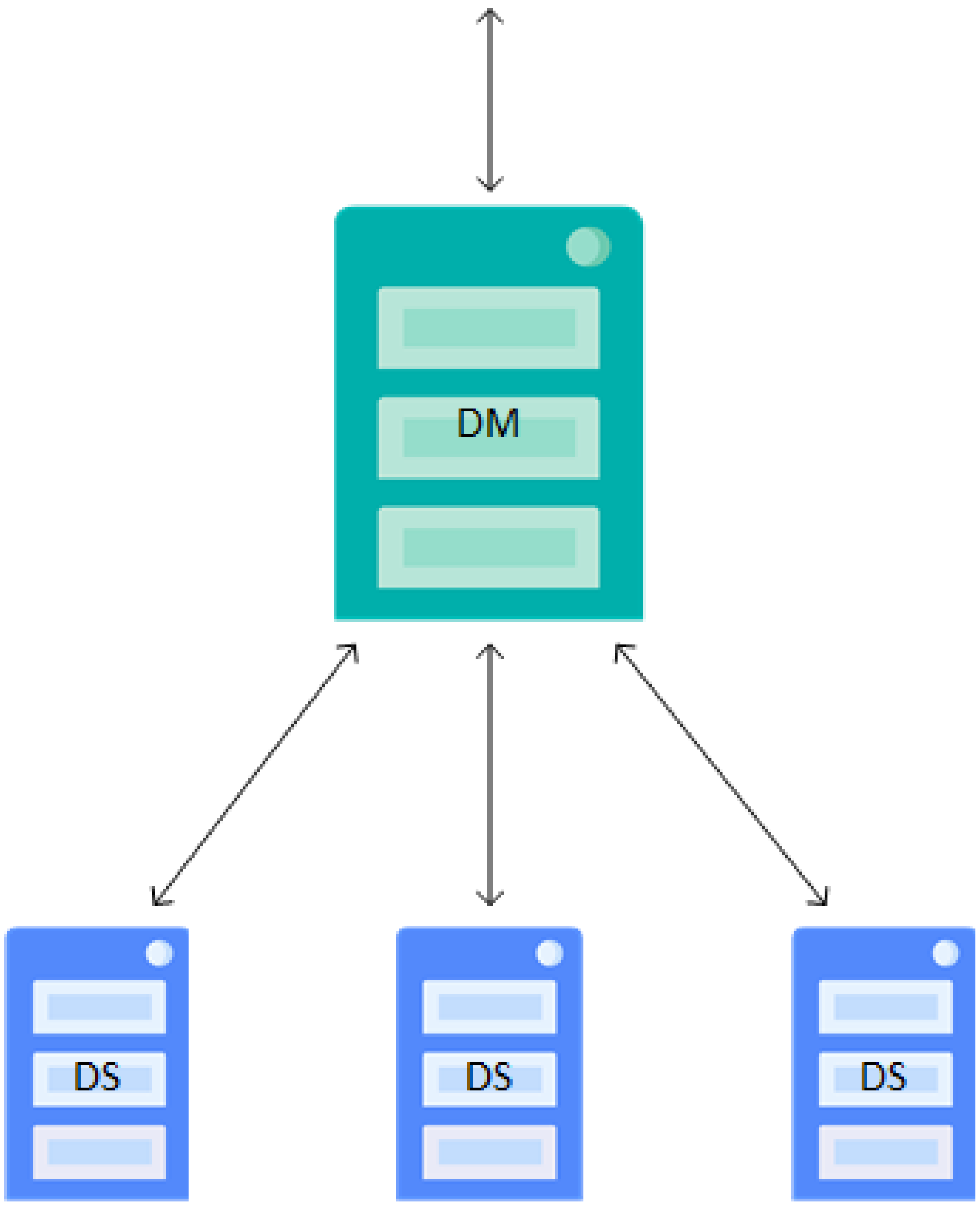
3.4.3 Sharded Cluster Definitions File

The definitions.json file for the sharded cluster configuration must define the following nodes:

- One shard master data server (role DM).
- Three shard data servers (role DS).

These are illustrated in the following:

Figure 3–2: Sharded Cluster to be Deployed by ICM



The table that follows lists the field/value pairs that are required for this configuration.

Requirement	Definition	Field: Value
Nodes for target InterSystems IRIS configuration, including provider-specific characteristics	Shard master data server using an InterSystems IRIS sharding license. Instance type overrides defaults file due to database cache requirements. Database cache size specified (8KB block size). Data volume size overrides defaults file.	"Role": "DM" "LicenseKey": "sharding-iris.key” "InstanceType": "m4.4xlarge" "ISCglobals": "0,0,40,960,0,0,0" DataVolumeSize": "60"
	+16178187401@mymetropcs.com	"Role": "DS" "Count": "4" StartCount”: “2” "LicenseKey": "sharding-iris.key” "InstanceType": "m4.10xlarge" "ISCglobals": "0,0,143360,0,0,0" "DataVolumeSize": "115"

Note: For more detailed information about the specifics of deploying a sharded cluster, such as database cache size and data volume size requirements, see Deploying the Sharded Cluster in the “Horizontally Scaling InterSystems IRIS for Data Volume with Sharding” chapter of the *Scalability Guide*.

3.5 Provision the Infrastructure

ICM provisions cloud infrastructure using the HashiCorp Terraform tool.

- The icm provision Command
- Infrastructure Management Commands

Note: ICM gives you the option of provisioning your own existing cloud or virtual compute nodes or physical servers to deploy containers on; see Deploying on a Preexisting Cluster for more information.

3.5.1 The icm provision Command

The **icm provision** command allocates and configures compute nodes, using the field values provided in the definitions.json and defaults.json files, as well as default values for unspecified parameters where applicable. By default, the input files in the current working directory are used; you can specify another location using the **-definitions** and **-defaults** options. In the case of the separate definitions files for the two target configurations (see Define the Deployment), the appropriate file can be swapped in as definitions.json in the current working directory before the **icm provision** command is executed.

Note: If you use the **-definitions** or **-defaults** options to specify a nondefault location for one or both of these configuration files, you must also do so for all subsequent ICM commands you run for this deployment. For example, if you execute **icm provision -defaults ./config_files**, you must add **-defaults ./config_files** to all subsequent commands you issue for that deployment.

While the provisioning operation is ongoing, ICM provides status messages regarding the *plan* phase (the Terraform phase that validates the desired infrastructure and generates state files) and the *apply* phase (the Terraform phase that accesses the cloud provider, carries out allocation of the machines, and updates state files). Because ICM runs Terraform in multiple threads, the order in which machines are provisioned and in which additional actions applied to them is not deterministic. This is illustrated in the sample output that follows.

At completion, ICM also provides a summary of the compute nodes and associated components that have been provisioned, and outputs a command line which can be used to delete the infrastructure at a later date.

Important: Unprovisioning public cloud compute nodes in a timely manner avoids unnecessary expense. Because the **-stateDir** option to the **icm unprovision** command is mandatory, you may find it convenient to copy the **icm unprovision** command provided in the output, so you can easily replicate it when unprovisioning. This output also appears in the icm.log file.

The following example if excerpted from the output of provisioning of the distributed cache cluster described in Define the Deployment.

```
$ icm provision -definitions definitions_DCC.json
Starting init of ACME-TEST...
...completed init of ACME-TEST
Starting plan of ACME-DM-TEST...
...
Starting refresh of ACME-TEST...

...
Starting apply of ACME-DM-TEST...
...
Copying files to ACME-DM-TEST-0002...
...
Configuring ACME-AM-TEST-0003...
...
Mounting volumes on ACME-AM-TEST-0004...
...
Installing Docker on ACME-AM-TEST-0003...
...
Installing Weave Net on ACME-DM-TEST-0001...
...
Collecting Weave info for ACME-AR-TEST-0006...
...
...collected Weave info for ACME-AM-TEST-0005
...installed Weave Net on ACME-AM-TEST-0004

Machine            IP Address       DNS Name
-------             ---------        -------
ACME-DM-TEST-0001   00.53.183.209    ec2-00-53-183-209.us-west-1.compute.amazonaws.com
ACME-DM-TEST-0002   00.53.183.185    ec2-00-53-183-185.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003   00.56.59.42      ec2-00-56-59-42.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0005   00.67.1.11       ec2-00-67-1-11.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003   00.193.117.217   ec2-00-193-117-217.us-west-1.compute.amazonaws.com
ACME-LB-TEST-0002   (virtual AM)     ACME-AM-TEST-1546467861.amazonaws.com
ACME-AR-TEST-0006   00.53.201.194    ec2-00-53-201-194.us-west-1.compute.amazonaws.com
To destroy: icm unprovision -stateDir /Samples/AWS/ICM-8620265620732464265 [-cleanUp] [-force]
```

During the provisioning operation, ICM creates or updates state and log files in the state directory (created by ICM, with a name beginning with **ICM-**) and when finished creates the instances.json file, which serves as input to subsequent deployment and management commands. (See The Instances File in the chapter “Essential ICM Elements” for more information about this file.) By default, the instances file is created in the current working directory; you can change this using the **-instances** option, but note that if you do you must supply the alternate location by using the **-instances** option with all subsequent commands.

Because interactions with cloud providers sometimes involve high latency leading to timeouts and internal errors on the provider side, the **icm provision** command is fully *reentrant* to make the provisioning process as resilient as possible. If errors are encountered during provisioning, the command can be issued multiple times until ICM completes all the required tasks for all the specified nodes without error. When this happens, however, you must use the **-stateDir** option to specify the state directory (see The State Directory and State Files) in your repeated execution of the command, to indicate that provisioning is already in process and provide the needed information about what has been done and what hasn't. For example, suppose you encounter the problem in the following example:

```
$ icm provision
Starting plan of ACME-DM-TEST...
...completed plan of ACME-DM-TEST
Starting apply of ACME-AM-TEST...
Error: Thread exited with value 1
See /Samples/AWS/ICM-1105110161490759817/Sample-DS-TEST/terraform.err
To reprovision, specify --stateDir=/Samples/AWS/ICM-3078941885014382438
```

Review the indicated errors, fix as needed, then run **icm provision** again with the **-stateDir** option, as in the following

```
$ icm provision --stateDir=/Samples/AWS/ICM-3078941885014382438
Starting plan of ACME-DM-TEST...
...completed plan of ACME-DM-TEST
Starting apply of ACME-DM-TEST...
...completed apply of ACME-DM-TEST
[...]
To destroy: icm unprovision -stateDir /tmp/ICM-3078941885014382438 [-cleanUp] [-force]
```

Even when provisioning is successful, you can run the **icm provision** command again (with the **-stateDir** option) after making changes to your configuration files to alter the provisioned infrastructure. For example, you can change storage volume sizes for some of the nodes in the definitions file and execute **icm provision** again to reprovision with the new sizes.

By default, when issuing the **icm provision** command to modify existing infrastructure, ICM prompts you to confirm; you can avoid this by using the **-force** option, for example when using a script.

3.5.2 Infrastructure Management Commands

The commands in this section are used to manage the infrastructure you have provisioned using ICM.

3.5.2.1 icm inventory

The **icm inventory** command lists the provisioned nodes, as at the end of the provisioning output, based on the information in the instances.json file (see The Instances File in the chapter "Essential ICM Elements"). For example:

```
$ icm inventory
Machine            IP Address       DNS Name
-------            ----------       --------
ACME-DM-TEST-0001   00.53.183.209-   ec2-52-53-183-209.us-west-1.compute.amazonaws.com
ACME-DM-TEST-0002   00.53.183.185+   ec2-52-53-183-185.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003   00.56.59.42      ec2-13-56-59-42.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0005   00.67.1.11       ec2-54-67-1-11.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003   00.193.117.217   ec2-54-193-117-217.us-west-1.compute.amazonaws.com
ACME-LB-TEST-0002   (virtual AM)     ACME-AM-TEST-1546467861.amazonaws.com
ACME-AR-TEST-0006   00.53.201.194    ec2-52-53-201-194.us-west-1.compute.amazonaws.com
```

You can also use the **-machine** or **-role** options to filter by node name or role, for example, with the same cluster as in the preceding example:

```
$ icm inventory -role AM
Machine            IP Address       DNS Name
-------            ----------       --------
ACME-AM-TEST-0003   00.56.59.42      ec2-13-56-59-42.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0005   00.67.1.11       ec2-54-67-1-11.us-west-1.compute.amazonaws.com
ACME-AM-TEST-0003   00.193.117.217   ec2-54-193-117-217.us-west-1.compute.amazonaws.com
```

3.5.2.2 icm ssh

The **icm ssh** command runs an arbitrary command on the specified compute nodes. Because mixing output from multiple commands would be hard to interpret, the output is written to files and a list of output files provided, for example:

```
$ icm ssh -command "ping -c 5 intersystems.com" -role DM
Executing command 'ping -c 5 intersystems.com' on ACME-DM-TEST-0001...
Executing command 'ping -c 5 intersystems.com' on ACME-DM-TEST-0002...
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0001/ssh.out
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0002/ssh.out
```

However, when the **-machine** or **-role** options are used to specify exactly one node, as in the following, the output is also written to the console:

```
$ icm ssh -command "df -k" -machine ACME-DM-TEST-0001
Executing command 'df -k' on ACME-DM-TEST-0001...
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0001/ssh.out

Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs          10474496 2205468   8269028  22% /
tmpfs            3874116       0   3874116   0% /dev
tmpfs            3874116       0   3874116   0% /sys/fs/cgroup
/dev/xvda2      33542124 3766604  29775520  12% /host
/dev/xvdb       10190100   36888   9612540   1% /irissys/data
/dev/xvdc       10190100   36888   9612540   1% /irissys/wij
/dev/xvdd       10190100   36888   9612540   1% /irissys/journal1
/dev/xvde       10190100   36888   9612540   1% /irissys/journal2
shm                65536     492     65044   1% /dev/shm
```

The **icm ssh** command can also be used in interactive mode to execute long-running, blocking, or interactive commands on a compute node. Unless the command is run on a single-node deployment, the **-interactive** flag must be accompanied by a **-role** or **-machine** option restricting the command to a single node. If the **-command** option is not used, the destination user's default shell (for example **bash**) is launched.

See icm exec for an example of running a command interactively.

Note: Two commands described in Service Management Commands, **icm exec** (which runs an arbitrary command on the specified containers) and **icm session** (which opens an interactive session for the InterSystems IRIS instance on a specified node) can be grouped with **icm ssh** as a set of powerful tools for interacting with your ICM deployment.

3.5.2.3 icm scp

The **icm scp** command securely copies a file or directory from the local ICM container to the host OS of the specified node or nodes. The command syntax is as follows:

```
icm scp -localPath local-path [-remotePath remote-path]
```

Both *localPath* and *remotePath* can be either files or directories. If *remotePath* is a directory, it must contain a trailing forward slash (/), or it will be assumed to be a file. If both are directories, the contents of the local directory are recursively copied; if you want the directory itself to be copied, remove the trailing slash (/) from *localPath*.

The default for the optional *remote-path* argument is /home/*ssh-user*. The root directory of this path, /home, is the default home directory; to change it, specify a different root directory using the Home field. The user specified by the SSHUser field must have the needed permissions for *remotePath*.

Note: See also the **icm cp** command, which copies a local file or directory on the specified node into the specified container.

3.6 Deploy and Manage Services

ICM carries out deployment of software services using Docker images, which it runs as containers by making calls to Docker. Containerized deployment using images supports ease of use and DevOps adaptation while avoiding the risks of manual upgrade. In addition to Docker, ICM also carries out some InterSystems IRIS-specific configuration over JDBC.

There are many container management and orchestration tools available, and these can be used to extend ICM's deployment and management capabilities.

- The icm run Command
- Container Management Commands
- Service Management Commands

Note: This section describes deployment of containerized InterSystems IRIS and other services. However, ICM can also deploy noncontainerized InterSystems IRIS using installation kits; for more information, see the appendix "Containerless Deployment".

3.6.1 The icm run Command

The **icm run** command pulls, creates, and starts a container from the specified image on each of the provisioned nodes. By default, the image specified by the DockerImage field in the configuration files is used, and the name of the deployed container is **iris**. This name is reserved for and should be used only for containers created from the following InterSystems images (or images based on these images):

- **iris** — Contains an instance of InterSystems IRIS.
- **spark** — Contains an instance of InterSystems IRIS plus Apache Spark.

  The **spark** image allows you to conveniently add Apache Spark capabilities to an ICM-deployed sharded cluster. The deployed **spark**-based containers create a Spark framework corresponding to the deployment by starting Spark slaves on the DS nodes and a Spark master on the DM node, all preconfigured to connect to the InterSystems IRIS instances running in the containers. For more information on using Spark with InterSystems IRIS, see *Using the InterSystems IRIS Spark Connector*.

  Note: To start Spark in the deployed containers when using this image, you must include **"spark": "true"** in the defaults file.

- **arbiter** — Contains an ISCAgent instance to act as mirror arbiter.

  The **arbiter** image is deployed on an AR node, which is configured as the arbiter host in a mirrored deployment. For more information on mirrored deployment and topology, see ICM Cluster Topology.

- **webgateway** — Contains an InterSystems Web Gateway installation along with an Apache web server.

  The **webgateway** image is deployed on a WS node, which is configured as a web server for a DM or an AM node; see ICM Node Types.

By including the DockerImage field in each node definition in the definitions.json file, you can run different InterSystems IRIS images on different node types. For example, you must do this to run the **arbiter** image on the AR node and the **webgateway** image on WS nodes while running the **iris** image on the other nodes.

Important: When the DockerImage field specifies the **iris** or **spark** image in the defaults.json file and you include an AR or WS definition in the definitions.json file, you *must* use include the DockerImage field in the AR or WS definition to override the default and specify the appropriate image (**arbiter** or **webgateway**, respectively) and avoid configuration errors.

Docker images from InterSystems comply with the OCI support specification, and are supported on Docker Enterprise Edition and Community Edition 18.03 and later. The version of Docker installed on provisioned nodes by the ICM command can be specified using the DockerVersion parameter; for more information, see General Parameters.

You can also use the **-image** and **-container** command-line options with **icm run** to specify a different image and container name. This allows you to deploy multiple containers created from multiple images on each provisioned node by using the **icm run** command multiple times — the first time to run the images specified by the DockerImage fields in the node definitions and deploy the **iris** container (of which there can be only one) on each node, as described in the foregoing paragraphs, and one or more subsequent times with the **-image** and **-container** options to run a custom image on all of the nodes or some of the nodes. Each container running on a given node must have a unique name. The **-machine** and **-role** options can also be used to restrict container deployment to a particular node, or to nodes of a particular type, for example, when deploying your own custom container on a specific provisioned node.

Another frequently used option, **-iscPassword**, specifies the InterSystems IRIS password to set for all deployed InterSystems IRIS containers; this value could be included in the configuration files, but the command line option avoids committing a password to a plain-text record. If the InterSystems IRIS password is not provided by either method, ICM prompts for it (with typing masked).

Note: For security, ICM never transmits the InterSystems IRIS password (however specified) in plain text, but instead generates a hashed password and salt locally, then sends these using SSH to the deployed InterSystems IRIS containers on the compute nodes.

Given all of the preceding, consider the following three examples of container deployment using the **icm run** command. (These do not present complete procedures, but are limited to the procedural elements relating to the deployment of particular containers on particular nodes.)

- To deploy a distributed cache cluster with one DM node and several AM nodes:

  1. When creating the defaults.json file, as described in Configuration, State, and Log Files and Define the Deployment, include the following to specify the default image from which to create the **iris** containers:

     ```
     "DockerImage": "intersystems/iris:stable"
     ```

  2. Execute the following command on the ICM command line:

     ```
     icm run -iscPassword "<password>"
     ```

     A container named **iris** containing an InterSystems IRIS instance with its initial password set as specified is deployed on each of the nodes; ICM performs the needed ECP configuration following container deployment.

- To deploy a basic sharded cluster with a mirrored DM node and an AR (arbiter) node:

  1. When creating he defaults.json file, as described in Configuration, State, and Log Files and Define the Deployment, include the following to specify the default image from which to create the **iris** containers and to enable ICM to start Spark in the containers when deployed:

     ```
     "DockerImage": "intersystems/spark:stable"
     "Spark": "true"
     ```

     Note: You must also include **"Mirror": "True"** to enable the mirrored DM node, as described in Rules for Mirroring.

  2. When creating the definitions.json file, override the DockerImage field in the defaults file for the AR node only by specifying the **arbiter** image in the AR node definition, for example:

     ```
       {
          "Role": "AR",
          "Count": "1",
          "DockerImage": "intersystems/arbiter:stable"
        }
     ```

  3. Execute the following command on the ICM command line:

     ```
     icm run -iscPassword "<password>"
     ```

A container named **iris** containing both an InterSystems IRIS instance with its initial password set as specified and Apache Spark is deployed on each of the DM and DS nodes; a container named **iris** containing an ISCAgent to act as mirror arbiter is deployed on the AR node; ICM performs the needed sharded cluster, Spark, and mirroring configuration following container deployment.

- To deploy a DM node with a stand-alone InterSystems Iris instance in the **iris** container and an additional container created from a custom image, plus several WS nodes connected to the DM:

  1. When creating the definitions.json file, as described in Configuration, State, and Log Files and Define the Deployment, specify the **iris** image for the DM node and the **webgateway** image for the WS nodes, for example:

     ```
     {
          "Role": "DM",
          "Count": "1",
          "DockerImage": "intersystems/iris:stable"
      },
      {
          "Role": "WS",
          "Count": "3",
          "DockerImage": "intersystems/webgateway:stable"
      }
     ```

  2. Execute the following command on the ICM command line:

     ```
     icm run
     ```

     ICM prompts for the initial InterSystems IRIS password with typing masked, and a container named **iris** containing an InterSystems IRIS instance is deployed on the DM node, a container named **iris** containing an InterSystems Web Gateway installation and an Apache web server is deployed on each of the WS nodes, and ICM performs the needed web server configuration following container deployment.

  3. Execute an ICM command to deploy the custom container on the DM node, for example either of the following:

     ```
     icm run -container customsensors -image myrepo/sensors:stable -role DM
     ```

     ```
     icm run -container customsensors -image myrepo/sensors:stable -machine ACME-DM-TEST-0001
     ```

     A container named **customsensors** created from the image **sensors** in your repository is deployed on the DM node.

Bear in mind the following further considerations:

- The container name **iris** remains the default for all ICM container and service management commands (as described in the following sections), so when you execute a command involving an additional container you have deployed using another name, you must refer to that name explicitly using the **-container** option. For example, to remove the custom container in the last example from the DM node, you would issue the following command:

  ```
  icm rm -container customsensors -machine ACME-DM-TEST-0001
  ```

  Without **-container customsensors**, this command would remove the **iris** container by default.

- The DockerRegistry, DockerUsername, and DockerPassword fields are required to specify and log into (if it is private) the Docker repository in which the specified image is located; for details see Docker Repositories.

- If you use the **-namespace** command line option with the **icm run** command to override the namespace specified in the defaults file (or the default of USER if not specified in defaults), the value of the Namespace field in the instances.json file (see The Instances File in the chapter "Essential ICM Elements") is updated with the image you specified, and you must therefore also use the **-namespace** option to specify the created namespace when using the **icm session** and **icm sql** commands.

Additional Docker options, such as **--volume**, can be specified on the **icm run** command line using the **-options** option, for example:

```
icm run -options "--volume /shared:/host" image intersystems/iris:stable
```

For more information on the **-options** option, see Using ICM with Custom and Third-Party Containers.

The **-command** option can be used with **icm run** to provide arguments to (or in place of) the Docker entry point; for more information, see Overriding Default Commands.

Because ICM issues Docker commands in multiple threads, the order in which containers are deployed on nodes is not deterministic. This is illustrated in the example that follows.

Important: Unlike the **icm provision** command, the **icm run** command cannot simply be repeated if it fails on one or more nodes. Generally speaking, there are two causes for deployment failures.

- External factors such as network latency and disconnects or interruptions in cloud provider service.

  When deployment fails due to external factors, you must manually roll back deployment on all nodes before executing **icm run** again. The procedure is as follows:

  1. Enter **icm stop** on the ICM command line to stop any InterSystems IRIS containers that were successfully deployed.
  2. Enter **icm rm** to remove those containers.
  3. Execute the following command to remove InterSystems IRIS data from the storage volumes

     ```
     icm ssh -command "sudo rm -rf /intersys/*/*"
     ```

  4. Enter **icm run** again.

- One or more errors in the configuration files.

  When deployment fails due to configuration file error — for example, if you made a mistake typing the value for the DockerImage field — the needed correction to the configuration file may render the instances.json file created during provisioning inaccurate and thus invalid. To avoid further errors, after correcting the mistake you should unprovision the infrastructure and start again with the **icm provision** command, as described in the previous section.

The following example represents output from deployment of the sharded cluster configuration described in Define the Deployment. Repetitive lines are omitted for brevity.

```
$ icm run -definitions definitions_cluster.json
Executing command 'docker login' on ACME-DM-TEST-0001...
...output in /Samples/AWS/ICM-8620265620732464265/Sample-DM-TEST/ACME-DM-TEST-0001/docker.out
...
Pulling image intersystems/iris:stable on SHARD-DM-TEST-0001...
...pulled SHARD-DM-TEST-0001 image intersystems/iris:stable
...
Creating container iris on ACME-DS-TEST-0002...
...
Copying license directory /Samples/license/ to ACME-AM-TEST-0003...
...
Starting container iris on ACME-DS-TEST-0004...
...
Waiting for InterSystems IRIS to start on ACME-DS-TEST-0002...
...
Configuring SSL on ACME-DM-TEST-0001...
...
Enabling ECP on ACME-DS-TEST-0003...
...
Setting System Mode on ACME-DS-TEST-0002...
...
Acquiring license on ACME-DS-TEST-0002...
...
Enabling shard service on ACME-DM-TEST-0001...
...
```

```
Assigning shards on ACME-DM-TEST-0001...
...
Configuring application server on ACME-AM-TEST-0003...
...
Management Portal available at:
http://ec2-00-153-49-109.us-west-1.compute.amazonaws.com:52773/csp/sys/UtilHome.csp
```

At completion, ICM outputs a link to the Management Portal of the appropriate InterSystems IRIS instance. In this case, the provided Management Portal link is for the shard master data server running in the InterSystems IRIS container on ACME-DM-TEST-001.

3.6.2 Container Management Commands

The commands in this section are used to manage the containers you have deployed on your provisioned infrastructure.

3.6.2.1 icm ps

When deployment is complete, the **icm ps** command shows you the run state of containers running on the nodes, for example:

```
$ icm ps -container iris
Machine              IP Address      Container    Status   Health    Image
-------              ----------      ---------    ------   ------    -----
ACME-DS-TEST-0004    00.56.140.23    iris         Up       healthy   intersystems/iris:stable
ACME-DS-TEST-0003    00.53.190.37    iris         Up       healthy   intersystems/iris:stable
ACME-DS-TEST-0002    00.67.116.202   iris         Up       healthy   intersystems/iris:stable
ACME-DM-TEST-0001    00.153.49.109   iris         Up       healthy   intersystems/iris:stable
```

If the **-container** restriction is omitted, all containers running on the nodes are listed. This includes both other containers deployed by ICM (for example, Weave network containers, or any custom or third party containers you deployed using the **icm run** command) and any deployed by other means after completion of the ICM deployment..

Beyond node name, IP address, container name, and the image the container was created from, the **icm ps** command includes the following columns:

- **Status** — One of the following status values generated by Docker: **created**, **restarting**, **running**, **removing** (or **up**), **paused**, **exited**, or **dead**.
- **Health** — For **iris**, **arbiter**, and **webgateway** containers, one of the values **starting**, **healthy**, or **unhealthy**; for other containers **none** (or blank). When **Status** is **exited**, **Health** may display the exit value (where **0** means success).

  For **iris** containers the Health value reflects the health state of the InterSystems IRIS instance in the container. (For information about the InterSystems IRIS health state, see System Monitor Health State in the “Using the System Monitor” chapter of the *Monitoring Guide*). For **arbiter** containers it reflects the status of the ISCAgent, and for **webgateway** containers the status of the InterSystems Web Gateway web server. Bear in mind that **unhealthy** may be temporary, as it can result from a warning that is subsequently cleared.
- **Mirror** — When mirroring is enabled (see Rules for Mirroring), the mirror member status (for example **PRIMARY**, **BACKUP**, **SYNCHRONIZING**) returned by the **%SYSTEM.Mirror.GetMemberStatus()**mirroring API call. For example:

  ```
  $ icm ps -container iris
  Machine                  IP Address      Container    Status   Health    Mirror   Image
  -------                  ----------      ---------    ------   ------    ------   -----
  ACME-DM-TEST-0001    00.153.49.109   iris        Up      healthy   BACKUP   intersystems/iris:stable
  ACME-DM-TEST-0001    00.153.49.109   iris        Up      healthy   PRIMARY  intersystems/iris:stable
  ```

  For an explanation of the meaning of each status, see Mirror Member Journal Transfer and Dejournaling Status in the “Mirroring” chapter of the *High Availability Guide*.

Additional deployment and management phase commands are listed in the following. For complete information about these commands, see ICM Reference.

3.6.2.2 icm stop

The **icm stop** command stops the specified containers (or **iris** by default) on the specified nodes, or on all nodes if no machine or role constraints provided). For example, to stop the InterSystems IRIS containers on the application servers in the distributed cache cluster configuration:

```
$ icm stop -container iris -role DS

Stopping container iris on ACME-DS-TEST-0002...
Stopping container iris on ACME-DS-TEST-0004...
Stopping container iris on ACME-DS-TEST-0003...
...completed stop of container iris on ACME-DS-TEST-0004
...completed stop of container iris on ACME-DS-TEST-0002
...completed stop of container iris on ACME-DS-TEST-0003
```

3.6.2.3 icm start

The **icm start** command starts the specified containers (or **iris** by default) on the specified nodes, or on all nodes if no machine or role constraints provided). For example, to restart one of the stopped application server InterSystems IRIS containers:

```
$ icm start -container iris -machine ACME-DS-TEST-0002...
Starting container iris on ACME-DS-TEST-0002...
...completed start of container iris on ACME-DS-TEST-0002
```

3.6.2.4 icm pull

The **icm pull** command downloads the specified image to the specified machines. For example, to add an image to the shard master data server in the sharded cluster:

```
$ icm pull -image intersystems/webgateway:stable -role DM
Pulling ACME-DM-TEST-0001 image intersystems/webgateway:stable...
...pulled ACME-DM-TEST-0001 image intersystems/webgateway:stable
```

Note that the **-image** option is not required if the image you want to pull is the one specified by the **DockerImage** field in the definitions file, for example:

```
"DockerImage": "intersystems/iris:stable",
```

Although the **icm run** automatically command pulls any images not already present on the host, an explicit **icm pull** might be desirable for testing, staging, or other purposes.

3.6.2.5 icm rm

The **icm rm** command deletes the specified container (or **iris** by default), but not the image from which it was started, from the specified nodes, or from all nodes if no machine or role is specified. Only a stopped container can be deleted.

3.6.2.6 icm upgrade

The **icm upgrade** command replaces the specified container on the specified machines. ICM orchestrates the following sequence of events to carry out an upgrade:

1. Pull the new image
2. Create the new container
3. Stop the existing container
4. Remove the existing container
5. Start the new container

By staging the new image in steps 1 and 2, the downtime required between steps 3-5 is kept relatively short.

For example, to upgrade the InterSystems IRIS container on the shard master data server:

```
$ icm upgrade -image intersystems/iris:latest -machine ACME-AM-TEST-0003
Pulling ACME-AM-TEST-0003 image intersystems/iris:latest...
...pulled ACME-AM-TEST-0003 image intersystems/iris:latest
Stopping container ACME-AM-TEST-0003...
...completed stop of container ACME-AM-TEST-0003
Removing container ACME-AM-TEST-0003...
...removed container ACME-AM-TEST-0003
Running image intersystems/iris:latest in container ACME-AM-TEST-0003...
...running image intersystems/iris:latest in container ACME-AM-TEST-0003
```

The **-image** option is required for the **icm upgrade** command. When the upgrade is complete, the value of the DockerImage field in the instances.json file (see The Instances File in the chapter "Essential ICM Elements") is updated with the image you specified.

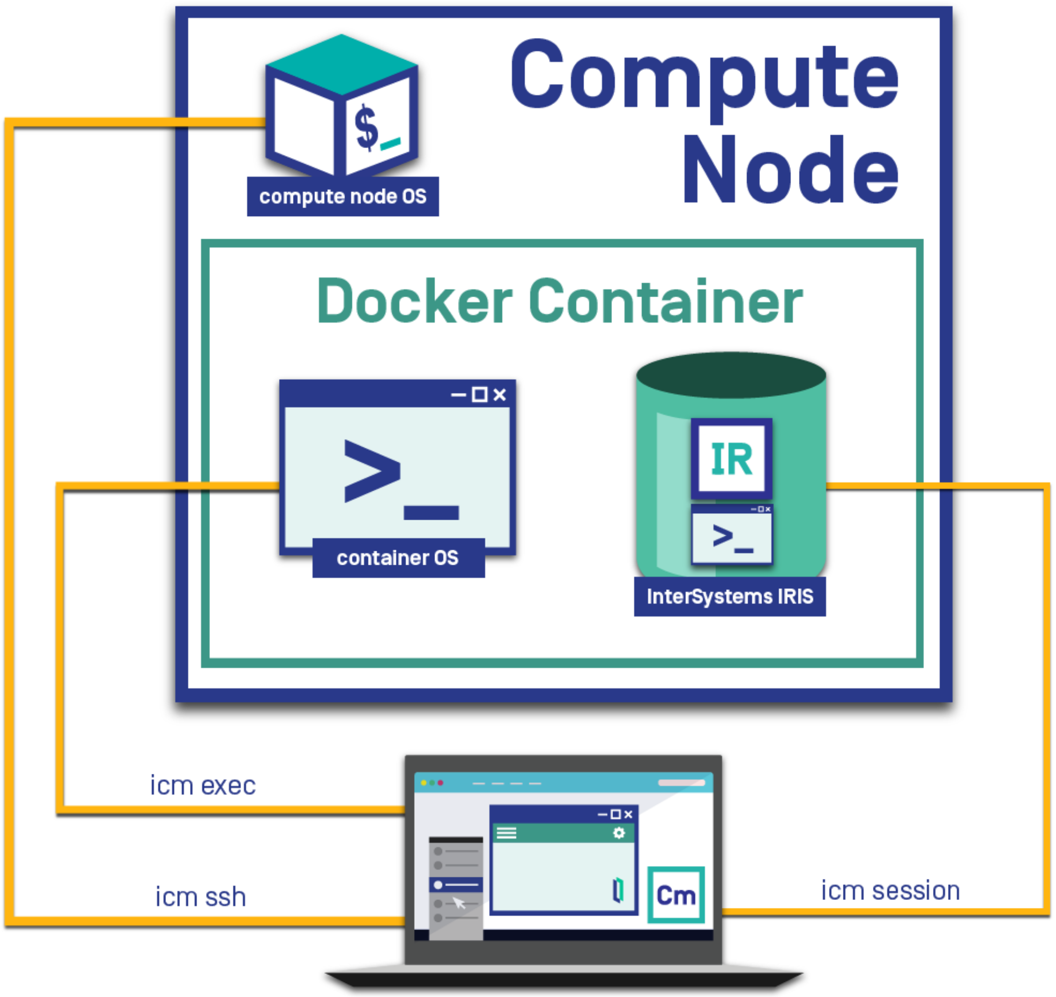
If you are upgrading a container other than **iris**, you must use the **-container** option to specify the container name.

3.6.3 Service Management Commands

These commands let you interact with the services running in your deployed containers, including InterSystems IRIS.

A significant feature of ICM is the ability it provides to interact with the nodes of your deployment on several levels — with the node itself, with the container deployed on it, and with the running InterSystems IRIS instance inside the container. The **icm ssh** (described in Infrastructure Management Commands), which lets you run a command on the specified compute nodes, can be grouped with the first two commands described in this section, **icm exec** (run a command in the specified containers) and **icm session** (open an interactive session for the InterSystems IRIS instance on a specified node) as a set of powerful tools for interacting with your ICM deployment. These multiple levels of interaction are shown in the following illustration.

Figure 3–3: Interactive ICM Commands



3.6.3.1 icm exec

The **icm exec** command runs an arbitrary command in the specified containers, for example

```
$ icm exec -command "df -k" -machine ACME-DM-TEST-0001
Executing command in container iris on ACME-DM-TEST-0001
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0001/docker.out

Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs          10474496 2205468   8269028  22% /
tmpfs            3874116       0   3874116   0% /dev
tmpfs            3874116       0   3874116   0% /sys/fs/cgroup
/dev/xvda2      33542124 3766604  29775520  12% /host
/dev/xvdb       10190100   36888   9612540   1% /irissys/data
/dev/xvdc       10190100   36888   9612540   1% /irissys/wij
/dev/xvdd       10190100   36888   9612540   1% /irissys/journal1
/dev/xvde       10190100   36888   9612540   1% /irissys/journal2
shm                65536     492     65044   1% /dev/shm
```

Because mixing output from multiple commands would be hard to interpret, when the command is executed on more than one node, the output is written to files and a list of output files provided.

Additional Docker options, such as **--env**, can be specified on the **icm exec** command line using the **-options** option; for more information on the **-options** option, see Using ICM with Custom and Third-Party Containers.

Because executing long-running, blocking, or interactive commands within a container can cause ICM to time out waiting for the command to complete or for user input, the **icm exec** command can also be used in interactive mode. Unless the command is run on a single-node deployment, the **-interactive** flag must be accompanied by a **-role** or **-machine** option restricting the command to a single node. A good example is running a shell in the container:

```
$ icm exec -command bash -machine ACME-AM-TEST-0004 -interactive
Executing command 'bash' in container iris on ACME-AM-TEST-0004...
[root@localhost /] $ whoami
root
[root@localhost /] $ hostname
iris-ACME-AM-TEST-0004
[root@localhost /] $ exit
```

Another example of a command to execute interactively within a container is an InterSystems IRIS command that prompts for user input, for example **iris stop**: which asks whether to broadcast a message before shutting down the InterSystems IRIS instance.

3.6.3.2 icm session

When used with the **-interactive** option, the **icm session** command opens an interactive session for the InterSystems IRIS instance on the node you specify. The **-namespace** option can be used to specify the namespace in which the session starts. For example:

```
$ icm session -interactive -machine ACME-AM-TEST-0003 -namespace %SYS

Node: iris-ACME-AM-TEST-0003, Instance: IRIS

Username: _SYSTEM
Password: ********
%SYS>
```

You can also use the **-command** option to provide a routine to be run in the InterSystems IRIS session, for example:

```
icm session -interactive -machine ACME-AM-TEST-0003 -namespace %SYS -command ^MIRROR
```

Additional Docker options, such as **--env**, can be specified on the **icm exec** command line using the **-options** option; for more information on the **-options** option, see Using ICM with Custom and Third-Party Containers.

Without the **-interactive** option, the **icm session** command runs the InterSystems IRIS ObjectScriptScript snippet specified by the **-command** option on the specified node or nodes. The **-namespace** option can be used to specify the namespace in which the snippet runs. Because mixing output from multiple commands would be hard to interpret, when the command is executed on more than one node, the output is written to files and a list of output files provided. For example:

```
$ icm session -command 'Write ##class(%File).Exists("test.txt")' -role AM
Executing command in container iris on ACME-AM-TEST-0003...
Executing command in container iris on ACME-AM-TEST-0004...
Executing command in container iris on ACME-AM-TEST-0005...
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-AM-TEST-0003/ssh.out
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-AM-TEST-0004/ssh.out
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-AM-TEST-0005/ssh.out
```

When the specified **-machine** or **-role** options limit the command to a single node, output is also written to the console, for example

```
$ icm session -command 'Write ##class(%File).Exists("test.txt")' -role DM
Executing command in container iris on ACME-DM-TEST-0001
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0001/docker.out

0
```

3.6.3.3 icm cp

The **icm cp** command copies a local file or directory on the specified node into the specified container. The command syntax is as follows:

```
icm cp -localPath local-path [-remotePath remote-path]
```

Both *localPath* and *remotePath* can be either files or directories. If both are directories, the contents of the local directory are recursively copied; if you want the directory itself to be copied, include it in *remotePath*.

The *remotePath* argument is optional and if omitted defaults to /tmp; if *remotePath* is a directory, it must contain a trailing forward slash (/), or it will be assumed to be a file. You can use the **-container** option to copy to a container other than the default **iris**.

Note: See also the **icm scp** command, which securely copies a file or directory from the local ICM container to the specified host OS.

3.6.3.4 icm sql

The **icm sql** command runs an arbitrary SQL command against the containerized InterSystems IRIS instance on the specified node (or all nodes), for example:

```
$ icm sql -command "SELECT Name,SMSGateway FROM %SYS.PhoneProviders" -role DM
Executing command in container iris on ACME-DM-TEST-0001...
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0001/jdbc.out

Name,SMSGateway
AT&amp;T Wireless,txt.att.net
Alltel,message.alltel.com
Cellular One,mobile.celloneusa.com
Nextel,messaging.nextel.com
Sprint PCS,messaging.sprintpcs.com
T-Mobile,tmomail.net
Verizon,vtext.com
```

The **-namespace** option can be used to specify the namespace in which the SQL command runs.

Because mixing output from multiple commands would be hard to interpret, when the command is executed on more than one node, the output is written to files and a list of output files provided.

3.6.3.5 icm docker

The **icm docker** command runs a Docker command on the specified node (or all nodes), for example:

```
$ icm docker -command "status --no-stream" -machine ACME-DM-TEST-0002
Executing command 'status --no-stream' on ACME-DM-TEST-0002...
...output in ./ICM-4780136574/ACME-DM-TEST/ACME-DM-TEST-0002/docker.out

CONTAINER     CPU %  MEM USAGE / LIMIT  MEM %  NET I/O     BLOCK I/O        PIDS
3e94c3b20340  0.01%  606.9MiB/7.389GiB  8.02%  5.6B/3.5kB  464.5MB/21.79MB  0
1952342e3b6b  0.10%  22.17MiB/7.389GiB  0.29%  0B/0B       13.72MB/0B       0
d3bb3f9a756c  0.10%  40.54MiB/7.389GiB  0.54%  0B/0B       38.43MB/0B       0
46b263cb3799  0.14%  56.61MiB/7.389GiB  0.75%  0B/0B       19.32MB/231.9kB  0
```

The Docker command should not be long-running (or block), otherwise control will not return to ICM. For example, if the **---no-stream** option in the example is removed, the call will not return until a timeout has expired.

3.7 Unprovision the Infrastructure

Because public cloud platform instances continually generate charges and unused instances in private clouds consume resources to no purpose, it is important to unprovision infrastructure in a timely manner.

The **icm unprovision** command deallocates the provisioned infrastructure based on the state files created during provisioning. The **-stateDir** option is required. As described in Provision the Infrastructure, **destroy** refers to the Terraform phase that deallocates the infrastructure. One line is created for each entry in the definitions file, regardless of how many nodes of that type were provisioned. Because ICM runs Terraform in multiple threads, the order in which machines are unprovisioned is not deterministic.

```
$ icm unprovision -stateDir /Samples/AWS/ICM-2416821167214483124 -cleanUp
Type "yes" to confirm: yes
Starting destroy of ACME-DM-TEST...
Starting destroy of ACME-AM-TEST...
Starting destroy of ACME-AR-TEST...
...completed destroy of ACME-AR-TEST
...completed destroy of ACME-AM-TEST
...completed destroy of ACME-DM-TEST
Starting destroy of ACME-TEST...
...completed destroy of Binstock-TEST
```

The **-cleanUp** option deletes the state directory after unprovisioning; by default, the state directory is preserved. The icm unprovision command prompts you to confirm unprovisioning by default; you can use the **-force** option to avoid this, for example when using a script.

4

ICM Reference

The following topics provide detailed information about various aspects of ICM and its use:

- ICM Commands and Options
- ICM Node Types
- ICM Cluster Topology and Mirroring
- Storage Volumes Mounted by ICM
- InterSystems IRIS Licensing for ICM
- ICM Security
- Deploying Across Multiple Zones
- Monitoring in ICM
- ICM Troubleshooting
- ICM Configuration Parameters

4.1 ICM Commands and Options

The first table that follows lists the commands that can be executed on the ICM command line; the second table lists the options that can be included with them. Both tables include links to relevant text.

Each of the commands is covered in detail in the “Using ICM” chapter. Command-line options can be used either to provide required or optional arguments to commands (for example, **icm exec -interactive**) or to set field values, overriding ICM defaults or settings in the configuration files (for example, **icm run -namespace "MIRROR1"**).

Note: The command table does not list every option that can be used with each command, and the option table does not list every command that can include each option.

Table 4–1: ICM Commands

Command	Description	Important Options
provi-sion	Provisions compute nodes	-definitions, -defaults, -instances

Command	Description	Important Options
inven-tory	Lists provisioned compute nodes	-machine, -role, -json
unprovi-sion	Destroys compute nodes	-stateDir, cleanup, force
ssh	Executes an operating system command on one or more compute nodes	-command, -machine, -role
scp	Copies a local file to one or more compute nodes	-localPath, -remotePath, -machine, -role
run	Deploys a container on compute nodes	-image, -container, -namespace, -options, -iscPassword, -command, -machine, -role
ps	Displays run states of containers deployed on compute nodes	-container, -json
stop	Stops containers on one or more compute nodes	-container, -machine, -role
start	Starts containers on one or more compute nodes	-container, -machine, -role
pull	Downloads an image to one or more compute nodes	-image, -container, -machine, -role
rm	Deletes containers from one or more compute nodes	-container, -machine, -role
upgrade	Replaces containers on one or more compute nodes	-image, -container, -machine, -role
exec	Executes an operating system command in one or more containers	-container, -command, -interactive. -options, -machine, -role
session	Opens an interactive session for an InterSys-tems IRIS instance in a container or executes an InterSystems IRIS ObjectScriptScript snippet on one or more instances	-namespace, -command, -interactive, , -options, -machine, -role
cp	Copies a local file to one or more containers	-localPath, -remotePath, -machine, -role
sql	Executes a SQL statement on the InterSystems IRIS instance	-namespace, -command, -machine, -role
docker	Executes a Docker command on one or more compute nodes	-container, -machine, -role

Table 4–2: ICM Command-Line Options

Option	Meaning	Default	Described in
-help	Display command usage information and ICM version		---
-version	Display ICM version		---
-verbose	Show execution detail	False	(can be used with any command)
-definitions *filepath*	Compute node definitions file	./definitions.json	Configuration, State and Log Files The icm provision Command
-defaults *filepath*	Compute node defaults file	./defaults.json	
-instances *filepath*	Compute node instances file	./instances.json	
-stateDir *dir*	Machine state directory	OS-specific	The State Directory and State Files The icm provision Command Unprovision the Infrastructure
-force	Don't confirm before reprovisioning or unprovisioning	False	The icm provision Command Unprovision the Infrastructure
-cleanUp	Delete *state direcorty* after unprovisioning	False	Unprovision the Infrastructure
-machine *regexp*	Target machine name pattern match	(all)	icm inventory icm ssh icm run icm exec icm session
-role *role*	Role of the InterSystems IRIS instance or instances for which a command is run, for example DM or QS	(all)	icm inventory icm ssh icm run icm exec
-namespace *namespace*	Namespace to create on deployed InterSystems IRIS instances and set as default execution namespace for the **session** and **sql** commands	USER	The Definitions File icm run icm session icm sql
-image *image*	Docker image to deploy; must include repository name.	DockerImage value in definitions file	icm run icm upgrade

Option	Meaning	Default	Described in
-options *options*	Additional Docker options	none	icm run icm exec icm session Using ICM with Custom and Third-Party Containers
-container *name*	Name of the container	**icm ps** command: (all) other commands: iris	icm run
-command *cmd*	Command or query to execute	none	icm ssh icm run icm exec icm session icm sql
-interactive	Redirect input/output to console for the **exec** and **ssh** commands	False	icm ssh icm exec
-localPath *path*	Local file or directory	none	icm scp icm cp
-remotePath *path*	Remote file or directory	/home/*SSHUser* (value of SSHUser field)	
-iscPassword *password*	Password for deployed InterSystems IRIS instances	iscPassword value in configuration file	icm run
-json	Enable JSON response mode	False	Using JSON Mode

Important: Use of the **-verbose** option, which is intended for debugging purposes only, may expose the value of iscPassword and other sensitive information, such as DockerPassword. When you use this option, you must either use the **-force** option as well or confirm that you want to use verbose mode before continuing.

4.2 ICM Node Types

This section described the types of nodes that can be provisioned and deployed by ICM and their possible roles in the deployed InterSystems IRIS configuration. A provisioned node's type is determined by the **Role** field.

The following table summarizes the detailed node type descriptions that follow it.

Node Type	Configuration Role(s)
DM	Shard master data server Distributed cache cluster data server Stand-alone InterSystems IRIS instance
AM	Shard master application server Distributed cache cluster application server
DS	Shard data server
QS	Shard query server
AR	Mirror arbiter
LB	Load balancer
WS	Web server
VM	Virtual machine
CN	Custom and third-party container node

4.2.1 Role DM: Shard Master Data Server, Distributed Cache Cluster Data Server, Standalone Instance

In an InterSystems IRIS sharded cluster (see the chapter "Horizontally Scaling InterSystems IRIS for Data Volume with Sharding" in the *Scalability Guide*), the shard master provides application access to the data shards on which the sharded data is stored and hosts nonsharded tables. (If shard master application servers [role AM] are included in the cluster, they provide application access instead.)

The node hosting the shard master is called the shard master data server. A shard master data server can be mirrored by deploying two nodes of role DM and specifying mirroring. The InterSystems IRIS Management Portal is typically accessed on the shard master data server; ICM provides a link to the portal on this node at the end of the deployment phase.

If multiple nodes of role AM and a DM node (nonmirrored or mirrored) are specified without any nodes of role DS (shard data server), they are deployed as an InterSystems IRIS distributed cache cluster, with the former serving as application servers and the latter as an data server.

Finally, a node of role DM (nonmirrored or mirrored) deployed by itself becomes a standalone InterSystems IRIS instance.

4.2.2 Role AM: Shard Master Application Server, Distributed Cache Cluster Application Server

When included in a sharded cluster, shard master application servers provide application access to the sharded data, distributing the user load across multiple nodes just as application servers in a distributed cache cluster do. If the shard master data server is mirrored, two or more shard master application servers must be included.

SQL and InterSystems IRIS ObjectScript commands that can be issued against the shard master data server (using the icm sql and icm session commands) can be issued against any of the shard master application servers. Shard master application servers automatically redirect application connections when a mirrored shard master data server fails over.

If multiple nodes of role AM and a DM node are specified without any nodes of role DS (shard data server), they are deployed as an InterSystems IRIS distributed cache cluster, with the former serving as application servers and the latter as an data server. When the data server is mirrored, application connection redirection following failover is automatic.

4.2.3 Role DS: Shard Data Server

A data shard stores one horizontal partition of each sharded table loaded into a sharded cluster. A node hosting a data shard is called a shard data server. A cluster can have two or more shard data servers up to over 200.

Shard data servers can be mirrored by deploying an even number and specifying mirroring.

4.2.4 Role QS: Shard Query Server

Query shards provides query access to the data shards to which they are assigned, minimizing interference between query and data ingestion workloads and increasing the bandwidth of a sharded cluster for high volume multiuser query workloads. A node hosting a query shard is called a shard query server. If shard data servers are deployed they are assigned round-robin to the deployed shard data servers.

Shard query servers automatically redirect application connections when a mirrored shard data server fails over.

4.2.5 Role AR: Mirror Arbiter

When a shard master data server, a distributed cache cluster data server, a stand-alone InterSystems IRIS instance, or shard data servers are mirrored, deployment of an arbiter node to facilitate automatic failover is highly recommended. One arbiter node is sufficient for all of the mirrors in a cluster; multiple arbiters are not supported and are ignored by ICM, as are arbiter nodes in a nonmirrored cluster.

The AR node does not contain an InterSystems IRIS instance, using a different image to run an ISCAgent container. This **arbiter** image must be specified using the DockerImage field in the definitions file entry for the AR node; for more information, see The icm run Command.

For more information about the arbiter, see the “Mirroring” chapter of the *High Availability Guide*.

4.2.6 Role LB: Load Balancer

ICM automatically provisions a load balancer node when the provisioning platform is AWS, GCP, or Azure, and the definition of nodes of type AM or WS in the definitions file contains the following parameter:

```
"LoadBalancer": "true"
```

For a custom load balancer, additional parameters must be provided.

4.2.6.1 Predefined Load Balancer

For nodes of role LB, ICM configures the ports and protocols to be forwarded as well as the corresponding health checks. Queries can be executed against the deployed load balancer the same way one would against a shard master application server or distributed cache cluster application server.

To add a load balancer to the definition of AM or WS nodes, add the LoadBalancer field, for example:

```
{
    "Role": "AM",
    "Count": "2",
    "LoadBalancer": "true"
}
```

The following example illustrates the nodes that would be created and deployed given this definition:

```
$ icm inventory
Machine              IP Address       DNS Name
-------              ----------       --------
ACME-AM-TEST-0001     54.214.230.24    ec2-54-214-230-24.amazonaws.com
ACME-AM-TEST-0002     54.214.230.25    ec2-54-214-230-25.amazonaws.com
ACME-LB-TEST-0000     (virtual AM)     ACME-AM-TEST-1546467861.amazonaws.com
```

Queries against this cluster can be executed against the load balancer the same way they would be against the AM nodes servers.

Currently, a single automatically provisioned load balancer cannot serve multiple node types (for example, both web servers and application servers), so each requires its own load balancer. This does not preclude the user from manually creating a custom load balancer for the desired roles.

4.2.6.2 Generic Load Balancer

A load balancer can be added to VM (virtual machine) and CN (container) nodes by providing the following additional keys:

- ForwardProtocol
- ForwardPort
- HealthCheckProtocol
- HealthCheckPath
- HealthCheckPort

The following is an example:

```
{
    "Role": "VM",
    "Count": "2",
    "LoadBalancer": "true",
    "ForwardProtocol": "tcp",
    "ForwardPort": "443",
    "HealthCheckProtocol": "http",
    "HealthCheckPath": "/csp/status.cxw",
    "HealthCheckPort": "8080"
}
```

More information about these keys can be found in ICM Configuration Parameters.

Note: A load balancer does not require (or allow) an explicit entry in the definitions file.

Some cloud providers create a DNS name for the load balancer that resolves to multiple IP addresses; for this reason, the value displayed by the provider interface as **DNS Name** should be used. If a numeric IP address appears in the **DNS Name** column, it simply means that the given cloud provider assigns a unique IP address to their load balancer, but doesn't give it a DNS name.

Because the DNS name may not indicate to which resources a given load balancer applies, the values displayed under **IP Address** are used for this purpose.

For providers VMware and PreExisting, you may wish to deploy a custom or third-party load balancer.

4.2.7 Role WS: Web Server

A cluster may contain any number of web servers. Each web server node contains an InterSystems Web Gateway installation along with an Apache web server. ICM populates the remote server list in the InterSystems Web Gateway with all of the available AM nodes (shard master application servers or distributed cache cluster application servers). If no AM nodes are available, it instead uses the DM node (shard master data server or distributed cache cluster data server); if mirrored, a mirror-aware connection is created. Finally, communication between the web server and remote servers is configured to run in SSL/TLS mode.

The WS node does not contain an InterSystems IRIS instance, using a different image to run a Web Gateway container. As described in The icm run Command, the **webgateway** image can be specified by including the DockerImage field in the WS node definition in the definitions.json file, for example:

```
{
    "Role": "WS",
    "Count": "3",
    "DockerImage": "intersystems/webgateway:stable",
    "ApplicationPath": "/acme",
    "AlternativeServers": "LoadBalancing"
}
```

If the ApplicationPath field is provided, its value is used to create an application path for each instance of the Web Gateway. The default server for this application path is assigned round-robin across Web Gateway instances, with the remaining remote servers making up the alternative server pool. For example, if the preceding sample WS node definition were part of a deployment with three AM nodes, the assignments would be like the following:

Instance	Default Server	Alternative Servers
ACME-WS-TEST-0001	ACME-AM-TEST-0001	ACME-AM-TEST-0002, ACME-AM-TEST-0003
ACME-WS-TEST-0002	ACME-AM-TEST-0002	ACME-AM-TEST-0001, ACME-AM-TEST-0003
ACME-WS-TEST-0003	ACME-AM-TEST-0003	ACME-AM-TEST-0001, ACME-AM-TEST-0002

The AlternativeServers field determines how the Web Gateway distributes requests to its target server pool. Valid values are LoadBalancing (the default) and FailOver. This field has no effect if the ApplicationPath field is not specified.

For information about using the InterSystems Web Gateway, see the *Web Gateway Configuration Guide*.

4.2.8 Role VM: Virtual Machine Node

A cluster may contain any number of virtual machine nodes. A virtual machine node provides a means of allocating compute instances which do not have a predefined role within an InterSystems IRIS cluster. Docker is not installed on these nodes, though users are free to deploy whatever custom or third-party software (including Docker) they wish.

The following commands are supported on the virtual machine node:

- icm provision
- icm unprovision
- icm inventory
- icm ssh
- icm scp

A load balancer may be assigned to a VM node; see Role LB: Load Balancer.

4.2.9 Role CN: Container Node

A cluster may contain any number of container nodes. A container node is a general purpose node with Docker installed. You can deploy any custom and third-party containers you wish on a CN node, except InterSystems IRIS containers, which will not be deployed if specified. All ICM commands are supported for container nodes, but most will be filtered out unless they use the **-container** option to specify a container other than **iris**, or the either the **-role** or **-machine** option is used to limit the command to CN nodes (see ICM Commands and Options).

A load balancer may be assigned to a CN node; see Role LB: Load Balancer.

4.3 ICM Cluster Topology and Mirroring

ICM validates the node definitions in the definitions file to ensure they meet certain requirements; there are additional rules for mirrored configurations. Bear in mind that this validation does not include configurations that are not functionally optimal, for example a single AM node, a single WS, 10 DS with only one QS node or vice-versa, and so on.

In both nonmirrored and mirrored configurations, QS nodes are assigned to DS nodes in round-robin fashion. If AM and WS nodes are not both included, they are all bound to the DM; if they are both included, AM nodes are bound to the DM and WS nodes to the AM nodes.

- Rules for Mirroring
- Nonmirrored Configuration Requirements
- Mirrored Configuration Requirements

4.3.1 Rules for Mirroring

The recommended general best practice for sharded clusters is that either the shard master data server (DM node) and the shard data servers (DS nodes) are all mirrored, or that none are mirrored. This is reflected in the following ICM topology validation rules :

When the Mirror field is set to False in the defaults file (the default), mirroring is never configured, and provisioning fails if more than one DM node is specified in the definitions file.

When the Mirror field is set to True, mirroring is configured where possible, as follows:

- If two DM nodes are specified in the definitions file, they are configured as a mirror failover pair using the default MirrorMap value, **primary,backup**.
- If more than two DMs are specified, and the MirrorMap field in the node definition matches the number of nodes specified and indicates that those beyond the failover pair are disaster recovery (DR) asyncs, they are configured as a failover pair and the specified number of DR asyncs. For example, the following definition creates a mirror consisting of a failover pair and two DR asyncs:

  ```
  "Role": "DM",
  "Count": "4",
  "MirrorMap": "primary,backup,async,async"
  ```

  The number of DM nodes can also be less than the number of elements in MirrorMap; in the example above, changing Count to **2** would deploy a primary and backup, while making it **3** would deploy the failover pair and one async.

  All asyncs deployed by ICM are DR asyncs; reporting asyncs are not supported. Up to 14 asyncs can be included in a mirror. For information on mirror members and possible configurations, see Mirror Components in the “Mirroring” chapter of the *High Availability Guide*.
- If one DM is specified, provisioning fails.
- If an even number of DS nodes is specified, they are configured as mirror failover pairs, for example specifying six DS nodes deploys three mirrored shard data servers. Async members are not supported for DS mirrors.
- If an odd number of DS nodes is specified, provisioning fails.
- If more than one AR (arbiter) node is specified, provisioning fails.

To see the mirror member status of each node in a configuration when mirroring is enabled, use the **icm ps** command.

Note: There is no relationship between the order in which DM or DS nodes are provisioned or configured and their roles in a mirror. You can determine which member of each pair is the primary failover member and which the backup using the **icm inventory** command, the output of which indicates each primary with a **+** (plus) and each backup with a **-** (minus).

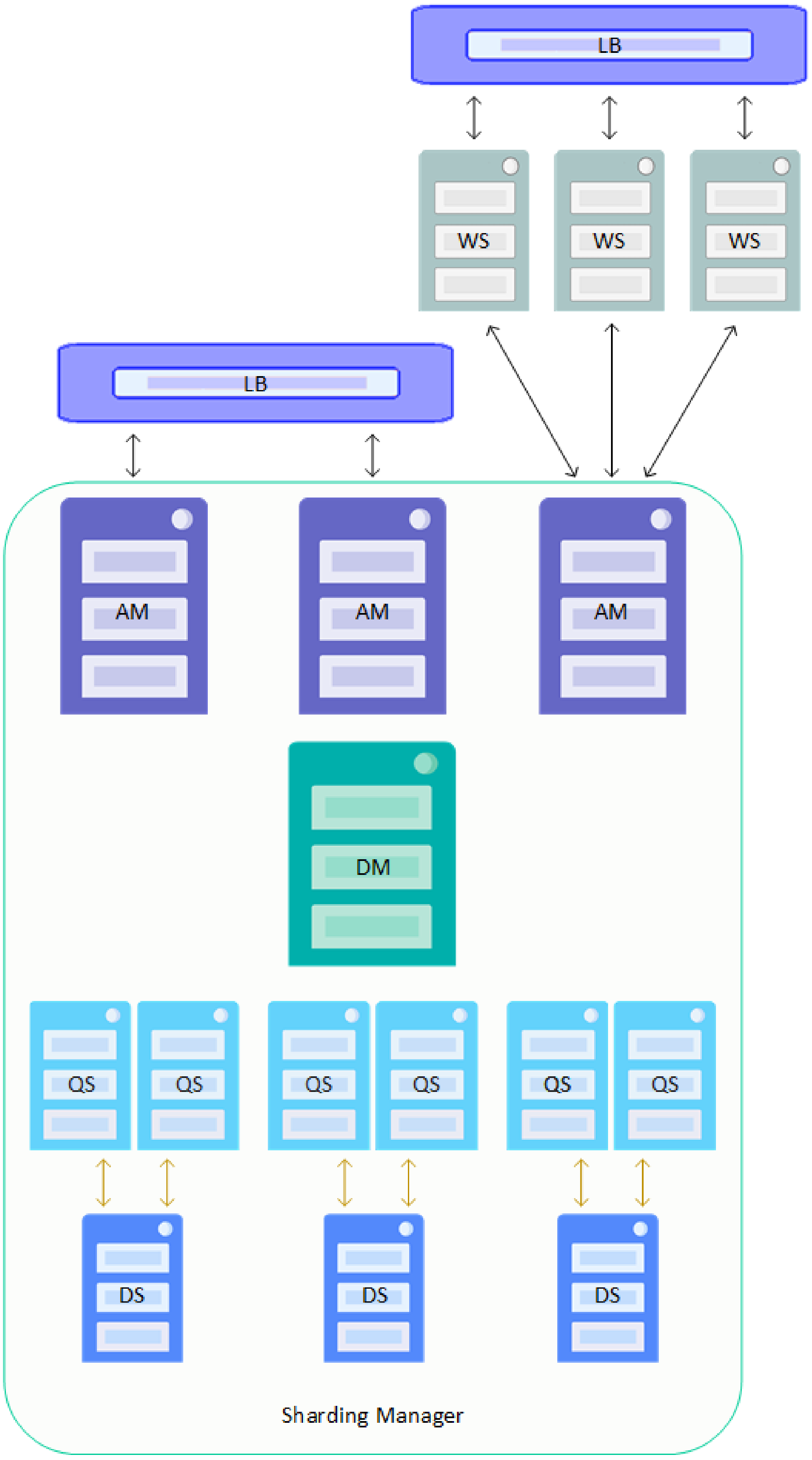
4.3.2 Nonmirrored Configuration Requirements

A non-mirrored cluster consists of the following:

- Exactly one DM (shard master data server, distributed cache cluster data server, standalone InterSystems IRIS instance)
- Zero or more AM (shard master application server, distributed cache cluster application server)
- Zero or more DS (shard data servers)
- Zero or more QS (shard query servers)
- Zero or more WS (web servers)
- Zero or more LB (load balancers)
- Zero AR (arbiter node is for mirrored configurations only)

The relationships between these nodes are pictured in the following illustration.

Figure 4–1: ICM Nonmirrored Topology



Note: The sharding manager, which handles communication between the shard master data server and the shard data servers in a basic cluster, enables direct connections between the shard master application servers and the shard query servers in this more complex configuration. For detailed information about connections within a sharded cluster, see the chapter "Horizontally Scaling InterSystems IRIS for Data Volume with Sharding" in the *Scalability Guide*.

4.3.3 Mirrored Configuration Requirements

A mirrored cluster consists of:

- Two DM (shard master data server, distributed cache cluster data server, standalone InterSystems IRIS instance) or more, if DR asyncs are specified by the MirrorMap field.
- Zero or more AM (shard master application server, distributed cache cluster application server)
- Even number of DS (shard data servers)
- Zero or more QS (shard query servers)
- Zero or more WS (web servers)
- Zero or more LB (load balancers)
- Zero or one AR (arbiter node is optional but recommended for mirrored configurations)

Note: This release of InterSystems IRIS does not support the use of async members in mirrors serving as shard master data server or shard data server nodes in sharded clusters.

A mirrored DM node that is deployed in the cloud without AM nodes must have some appropriate mechanism for redirecting application connections; see Redirecting Application Connections Following Failover or Disaster Recovery in the "Mirroring" chapter of the *High Availability Guide* for more information.

The following fields are required for mirroring:

- Mirroring is enabled by setting key Mirror in your defaults.json file to True.

  ```
  "Mirror": "True"
  ```

- To deploy more than two DM nodes, you must include the MirrorMap field in your definitions file to specify that those beyond the first two are DR async members, as follows:

  ```
  "MirrorMap": "primary,backup,async,..."
  ```

  The value of MirrorMap must always begin with **primary,backup**, and the number of elements in the field value must match the number of DM nodes, for example:

  ```
  "Role": "DM",
  "Count": "5",
  "MirrorMap": "primary,backup,async,async,async",
  ...
  ```

  MirrorMap can be used in conjunction with the Zone and ZoneMap fields to deploy async instances across zones; see Deploying Across Multiple Zones.

- Mirroring requires the choice of a namespace other than the default of USER. The namespace can be specified using the **-namespace** option with the icm run command or by including the Namespace field in your defaults file, as follows:

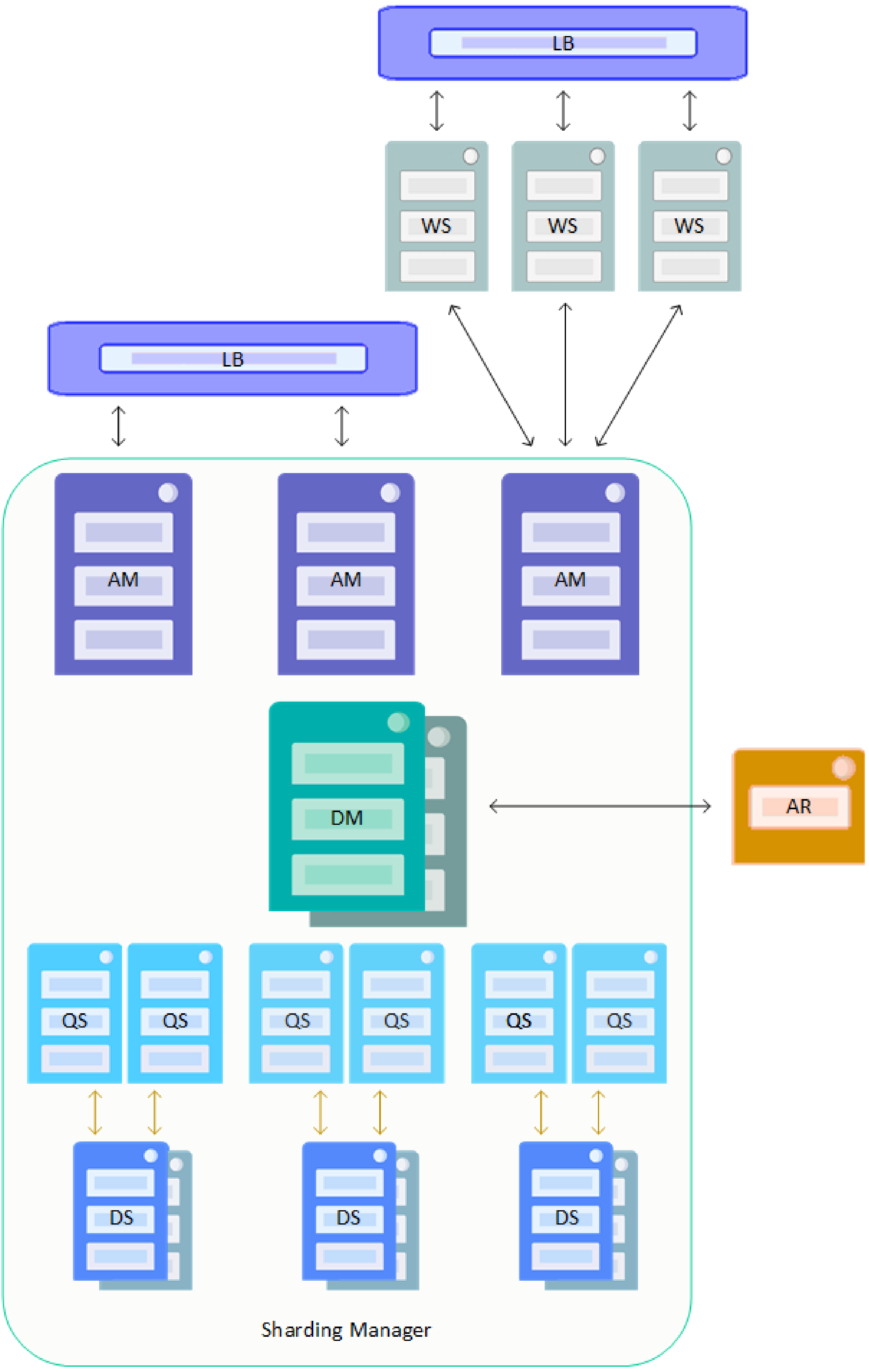
  ```
  "Namespace": "MIRROR1"
  ```

  Remote database bindings are assigned in the same fashion as a non-mirrored cluster, however the targets are mirrored pairs rather than individual servers.

Automatic LB deployment (see Role LB: Load Balancer) is supported for providers Amazon AWS, Microsoft Azure, and Google Cloud Platform; when creating your own load balancer, the pool of IP addresses to include are those of all AM and WS nodes.

The relationships between these nodes are pictured in the following illustration.

Figure 4–2: ICM Mirrored Topology



4.4 Storage Volumes Mounted by ICM

ICM must be able to format, partition, and mount volumes designated for use by InterSystems IRIS and Docker. The former are mounted by whatever InterSystems IRIS container is currently running on the host machine, with names determined by the fields DataDeviceName, WIJDeviceName, Journal1DeviceName, and Journal2DeviceName. The DockerDeviceName field specifies the name of the device mounted for use by Docker.

The devices must appear beneath /dev. For example, the following entry mounts the InterSystems IRIS data volume on /dev/sdb:

```
"DataDeviceName": "sdb"
```

For all providers other than type PreExisting, ICM attempts to assign reasonable defaults (see Device Name Parameters), but these values are highly platform and OS-specific and may need to be overridden in your defaults.json file. For PreExisting deployments, see Storage Volumes in the "Deploying on a Preexisting Cluster" appendix.

ICM mounts the InterSystems IRIS devices according to the following fields in your configuration files, the defaults for which are shown in the following:

Parameter	Default
DataMountPoint	/irissys/data
WIJMountPoint	/irissys/wij
Journal1MountPoint	/irissys/journal1
Journal2MountPoint	/irissys/journal2

4.5 InterSystems IRIS Licensing for ICM

InterSystems IRIS instances deployed in containers require licenses just as do noncontainerized instances. General InterSystems IRIS license elements and procedures are discussed in the "Licensing" chapter of the *System Administration Guide*.

License keys cannot be included in InterSystems IRIS container images, but must be added after the container is created and started. ICM addresses this as follows:

- The needed license keys are staged in a directory within the ICM container, or on a mounted volume, that is specified by the LicenseDir fields in the defaults.json file, for example /Samples/License.
- One of the license keys in the staging directory is specified by the LicenseKey field in each definition of node types DM, AM, DS, and QS in the definitions.json file, for example:

  ```
  "Role": "DM",
  "LicenseKey": "standard-iris.key",
  "InstanceType": "m4.xlarge",
  ...
  ```

- ICM configures a license server on the DM node (or nodes, if mirrored), which serves the specified licenses to the InterSystems IRIS nodes (including itself) during deployment.

All nodes in a sharded cluster require a sharding license. When deployed in nonsharded configurations, a standard license is sufficient for DM and AM nodes. No license is required for AR, LB, WS, VM, and CN nodes; if included in the definition for one of these, the LicenseKey field is ignored.

4.6 ICM Security

The security measures included in ICM are described in the following sections:

- Compute Node Communication
- Docker
- Weave Net
- Weave Scope
- InterSystems IRIS

For information about the ICM fields used to specify the files needed for the security described here, see Security-Related Parameters.

4.6.1 Compute Node Communication

This is the host machine on which containers are deployed. It may be virtual or physical, running in the cloud or on-premises.

ICM uses SSH to log into compute nodes and remotely execute commands on them, and SCP to copy files between the ICM container and a compute node. To enable this secure communication, you must provide an SSH public/private key pair and specify these keys in the defaults.json file as SSHPublicKey and SSHPrivateKey. During the configuration phase, ICM disables password login on each compute node, copies the private key to the node, and opens port 22, enabling clients with the corresponding public key to use SSH and SCP to connect to the node.

Other ports opened on the host machine are covered in the sections that follow.

4.6.2 Docker

During provisioning, ICM downloads and installs a specific version of Docker from the official Docker web site using a GPG fingerprint. ICM then copies the TLS certificates you provide (located in the directory specified by the TLSKeyDir field in the defaults file) to the host machine, starts the Docker daemon with TLS enabled, and opens port 2376. At this point clients with the corresponding certificates can issue Docker commands to the host machine.

4.6.3 Weave Net

During provisioning, ICM launches Weave Net with options to encrypt traffic and require a password (provided by the user) from each machine joining the Weave network.

Note: You can disable encryption of Weave Net traffic by setting the WeavePassword to the literal "null" in the defaults.json file. (By default, this parameter is generated by ICM and is set to the Weave Net password provided by the user.)

4.6.4 Weave Scope

ICM deploys Weave Scope with authentication enabled; credentials must be provided in the defaults.json file. For more information, see Monitoring in ICM.

4.6.5 InterSystems IRIS

For detailed and comprehensive information about InterSystems IRIS security, see the InterSystems IRIS Security Administration Guide.

4.6.5.1 Security Level

ICM expects that the InterSystems IRIS image was installed with Normal security (as opposed to Minimal or Locked Down).

4.6.5.2 Predefined Account Password

To secure the InterSystems IRIS instance, the default password for predefined accounts must be changed by ICM. The first time ICM runs the InterSystems IRIS container, passwords on all enabled accounts with non-null roles are changed to a password provided by the user. If you don’t want the InterSystems IRIS password to appear in the definitions files, or in your command-line history using the **-iscPassword** option, you can omit both; ICM interactively prompts for the password, masking your typing. Because passwords are persisted, they are not changed when the InterSystems IRIS container is restarted or upgraded.

4.6.5.3 JDBC

ICM opens JDBC connections to InterSystems IRIS in SSL/TLS mode (as required by InterSystems IRIS), using the files located in the directory specified by the TLSKeyDir field in the defaults file.

4.6.5.4 Mirroring

ICM creates mirrors (both DM pairs and DS failover pairs) with SSL/TLS enabled (see the “Mirroring" chapter of the *High Availability Guide*), using the files located in the directory specified by the TLSKeyDir field in the defaults file. Failover members can join a mirror only if SSL/TLS enabled.

4.6.5.5 InterSystems Web Gateway

ICM configures WS nodes to communicate with DM and AM nodes using SSL/TLS, using the files located in the directory specified by the TLSKeyDir field in the defaults file.

4.6.5.6 Centralized Security

InterSystems recommends the use of an LDAP server to implement centralized security across the nodes of a sharded cluster or other ICM deployment. For information about using LDAP with InterSystems IRIS, see the “Using LDAP” chapter of the *Security Administration Guide*.

4.7 Deploying Across Multiple Zones

Some cloud providers allow their virtual networks to span multiple zones within a given region. For some deployments, you may want to take advantage of this to deploy different nodes in different zones, For example, if you deploy a DM mirror that includes a failover pair and two DR asyncs (see Mirrored Configuration Requirements), you can accomplish the cloud equivalent of putting physical DR asyncs in remote data centers by deploying the failover pair, the first async, and the second async in three different zones.

To specify multiple zones when deploying on AWS, GCP, and Azure, populate the Zone field in the defaults file with a comma-separated list of zones. Here is an example for AWS:

```
{
    "Provider": "AWS",
    . . .
    "Region": "us-west-1",
    "Zone": "us-west-1b,us-west-1c"
}
```

For GCP:

```

    "Provider": "GCP",
    . . .
    "Region": "us-east1",
    "Zone": "us-east1-b,us-east1-c"
}
```

For Azure:

```
    "Provider": "Azure",
    . . .
    "Region": "Central US",
    "Zone": "1,2"
```

The specified zones are assigned to nodes in round-robin fashion. For example, if you use the first example and provision four DS nodes, the first and third will be provisioned in us-west-1b, the second and fourth in us-west-1c.

Round-robin distribution may lead to undesirable results, however; the preceding Zone specifications would place the primary and backup members of mirrored DM or DS nodes in different zones, for example, which might not be appropriate for your application due to higher latency between the members (see Network Latency Considerations in the “Mirroring” chapter of the *High Availability Guide*). To choose which nodes go in which zones, you can add the ZoneMaps field to a node definition in the definitions.json file to specify a particular zone specified by the Zone field for a single node or a pattern for zone placement for multiple nodes. This is shown in the following specifications for a distributed cache cluster with a mirrored data server:

- defaults.json

```
"Region": "us-west-1",
"Zone": "us-west-1a,us-west-1b,us-west-1c"
```

- definitions.json

```
"Role": "DM",
"Count": "4”,
"MirrorMap": "primary,backup,async,async",
"ZoneMap": "0,0,1,2",
...
"Role": "AM",
"Count": "3”,
"MirrorMap": "primary,backup,async,async",
"ZoneMap": "0,1,2",
...
"Role": "AR",
...
```

This places the primary and backup mirror members in us-west-1a and one application server in each zone, while the asyncs are in different zones from the failover pair to maximize their availability if needed — the first in us-west-1b and the second in us-west-1c. The arbiter node does not need a ZoneMap field to be placed in us-west-1a with the failover pair; round-robin distribution will take care of that.

4.8 Monitoring in ICM

ICM offers Weave Scope, a product of Weaveworks, as a basic monitoring facility. Weave Scope is not deployed by default, but must be specified in the defaults file using the Monitor field.

Weave Scope runs as a distributed system; there is no preferred node, and its web server is available on all compute instances (at port 4040). Because Weave Scope runs on top of Weave Net, it can be deployed only for Overlay Networks of type "weave". Weave can be deployed by including the following in your defaults.json file:

```
"Monitor": "scope"
```

Because the free version of Weave Scope does not offer authentication or HTTPS for its web interface, the ProxyImage parameter lets you specify an additional Docker image that ICM uses as a (reverse) proxy to the native Weave Scope interface, for example:

```
"ProxyImage": "intersystems/https-proxy-auth:stable"
```

The proxy configures HTTPS to use the SSL keys located in the directory specified by the TLSKeyDir parameter and carries out authentication using the MonitorUsername and MonitorPassword parameters.

When provisioning is complete, the port for the Weave Scope is displayed, for example:

```
Weave Scope available at https://54.191.23.24:4041
```

The Weave Scope UI is available at all node IP addresses, not just the one listed.

Note: Fully-qualified domain names may not work with unsigned certificates, in which case use the IP address instead.

4.9 ICM Troubleshooting

When an error occurs during an ICM operation, ICM displays a message directing you to the log file in which information about the error can be found. Before beginning an ICM deployment, familiarize yourself with the log files and their locations as described in Log Files and Other ICM Files.

In addition to the topics that follow, please see Additional Docker/InterSystems IRIS Considerations in *Running InterSystems IRIS in Containers* for information about important considerations when creating and running InterSystems IRIS images container images.

- Compute Node Restart and Recovery
- Correcting Time Skew
- Timeouts Under ICM
- Docker Bridge Network IP Address Range Conflict
- Weave Network IP Address Range Conflict
- Huge Pages

4.9.1 Compute Node Restart and Recovery

When a cloud compute node is shut down and restarted due to an unplanned outage or to planned action by the cloud provider (for example, for preventive maintenance) or user (for example, to reduce costs), its IP address and domain name may change, causing problems for both ICM and deployed applications (including InterSystems IRIS).

This behavior differs by cloud provider. GCP and Azure preserve IP address and domain name across compute node restart by default, whereas on AWS this feature is optional (see AWS Elastic IP Feature

Reasons a compute node might be shut down include the following:

- Unplanned outage
  - Power outage
  - Kernel panic
- Preventive maintenance initiated by provider
- Cost reduction strategy initiated by user

Methods for intentionally shutting down compute instances include:

- Using the cloud provider user interface
- Using ICM:

  ```
  icm ssh -command 'sudo shutdown'
  ```

4.9.1.1 AWS Elastic IP Feature

The AWS Elastic IP feature preserves IP addresses and domain names across compute node restarts. ICM disables this feature by default is because it incurs additional charges on stopped machines (but not running ones), and because AWS allows only five Elastic IP addresses per region (or VPC) unless a request is made to increase this limit. To enable this feature, set the ElasticIP field to True in your defaults.json file. For more information on this feature, see Elastic IP Addresses in the AWS documentation.

4.9.1.2 Recovery and Restart Procedure

If the IP address and domain name of a compute node change, ICM can no longer communicate with the node and a manual update is therefore required, followed by an update to the cluster. The procedure is as follows:

1. Go to the web console of the cloud provider and locate your instances there. Record the IP address and domain name of each, for example:

Node	IP Address	Domain Name
ACME-DM-TEST-0001	54.191.233.2	ec2-54-191-233-2.amazonaws.com
ACME-DS-TEST-0002	54.202.223.57	ec2-54-202-223-57.amazonaws.com
ACME-DS-TEST-0003	54.202.223.58	ec2-54-202-223-58.amazonaws.com

2. Edit the instances.json file (see The Instances File in the chapter “Essential ICM Elements”) and update the IPAddress and DNSName fields for each instance, for example:

```
"Label" : "ISC",
"Role" : "DM",
"Tag" : "TEST",
"MachineName" : "ACME-DM-TEST-0001",
"IPAddress" : "54.191.233.2",
"DNSName" : "ec2-54-191-233-2.amazonaws.com",
```

3. Verify that the values are correct using the icm inventory command:

```
$ icm inventory
Machine            IP Address       DNS Name
-------            ----------       --------
ACME-DM-TEST-0001   54.191.233.2     ec2-54-191-233-2.amazonaws.com
ACME-DS-TEST-0002   54.202.223.57    ec2-54-202-223-57.amazonaws.com
ACME-DS-TEST-0003   54.202.223.58    ec2-54-202-223-58.amazonaws.com
```

4. Use the icm ps command to verify that the compute instances are reachable:

```
$ icm ps -container weave
Machine            IP Address      Container   Status   Health    Image
-------            ----------      ---------   ------   ------    -----
ACME-DM-TEST-0001   54.191.233.2    weave       Up                 weaveworks/weave:2.0.4
ACME-DS-TEST-0002   54.202.223.57   weave       Up                 weaveworks/weave:2.0.4
ACME-DS-TEST-0003   54.202.223.58   weave       Up                 weaveworks/weave:2.0.4
```

5. The Weave network deployed by ICM includes a decentralized discovery service, which means that if at least one compute instance has kept its original IP address, the other compute instances will be able to reach it and reestablish all of their connections with one another. However, if the IP address of every compute instance in the cluster has changed, an additional step is needed to connect all the nodes in the Weave network to a valid IP address. Select one of the new IP addresses, such as **54.191.233.2** in our example. Then connect each node to this IP address using the icm ssh command, as follows:

```
$ icm ssh -command "weave connect --replace 54.191.233.2"
Executing command 'weave connect 54.191.233.2' on host ACME-DM-TEST-0001...
Executing command 'weave connect 54.191.233.2' on host ACME-DS-TEST-0002...
Executing command 'weave connect 54.191.233.2' on host ACME-DS-TEST-0003...
...executed on ACME-DM-TEST-0001
...executed on ACME-DS-TEST-0002
...executed on ACME-DS-TEST-0003
```

4.9.2 Correcting Time Skew

If the system time within the ICM containers differs from Standard Time by more than a few minutes, the various cloud providers may reject requests from ICM. This can happen if the container is unable to reach an NTP server on startup (initial or after being stopped or paused). The error appears in the terraform.err file as some variation on the following:

```
Error refreshing state: 1 error(s) occurred:

    # icm provision
    Error: Thread exited with value 1
    Signature expired: 20170504T170025Z is now earlier than 20170504T171441Z (20170504T172941Z   15
min.)
   status code: 403, request id: 41f1c4c3-30ef-11e7-afcb-3d4015da6526 doesn’t run for a period of time
```

The solution is to manually run NTP, for example:

```
ntpd -nqp pool.ntp.org
```

and verify that the time is now correct. (See also the discussion of the **--cap-add** option in Launch ICM.)

4.9.3 Timeouts Under ICM

When the target system is under extreme load, various operations in ICM may time out. Many of these timeouts are not under direct ICM control (for example, from cloud providers); other operations are retried several times, for example SSH and JDBC connections.

SSH timeouts are sometimes not identified as such. For instance, in the following example, an SSH timeout manifests as a generic exception from the underlying library:

```
# icm cp -localPath foo.txt -remotePath /tmp/
2017-03-28 18:40:19 ERROR Docker:324 - Error:
java.io.IOException: com.jcraft.jsch.JSchException: channel is not opened.
2017-03-28 18:40:19 ERROR Docker:24 - java.lang.Exception: Errors occurred during execution; aborting
 operation
        at com.intersystems.tbd.provision.SSH.sshCommand(SSH.java:419)
        at com.intersystems.tbd.provision.Provision.execute(Provision.java:173)
        at com.intersystems.tbd.provision.Main.main(Main.java:22)
```

In this case the recommended course of action is to retry the operation (after identifying and resolving its proximate cause).

Note that for security reasons ICM sets the default SSH timeout for idle sessions at ten minutes (60 seconds x 10 retries). These values can be changed by modifying the following fields in the/etc/ssh/sshd_config file:

```
ClientAliveInterval 60
ClientAliveCountMax 10
```

4.9.4 Docker Bridge Network IP Address Range Conflict

For container networking, Docker uses a bridge network (see Use bridge networks in the Docker documentation) on subnet 172.17.0.0/16 by default. If this subnet is already in use on your network, collisions may occur that prevent Docker from starting up or prevent you from being able to reach your deployed compute nodes. This problem can arise on the machine hosting your ICM container, your InterSystems IRIS cluster nodes, or both.

To resolve this, you can edit the bridge network’s IP configuration in the Docker configuration file to reassign the subnet to a range that is not in conflict with your own IP addresses (your IT department can help you determine this value). To make this change, add a line like the following to the Docker daemon configuration file:

```
"bip:" "192.168.0.1/24"
```

If the problem arises with the ICM container, edit the file /etc/docker/daemon.json on the container’s host. If the problem arises with the compute nodes in a deployed configuration, edit the file /ICM/etc/toHost/daemon.json in the ICM container; by default this file contains the value in the preceding example, which is likely to avoid problems with any deployment type except PreExisting.

Detailed information about the contents of the daemon.json file can be found in Daemon configuration file in the Docker documentation; see also Configure and troubleshoot the Docker daemon.

4.9.5 Weave Network IP Address Range Conflict

By default, the Weave network uses IP address range 10.32.0.0/12. If this conflicts with an existing network, you may see an error such as the following in log file installWeave.log:

```
Network 10.32.0.0/12 overlaps with existing route 10.0.0.0/8 on host
ERROR: Default --ipalloc-range 10.32.0.0/12 overlaps with existing route on host.
You must pick another range and set it on all hosts.
```

This is most likely to occur with provider PreExisting if the machines provided have undergone custom network configuration to support other software or local policies. If disabling or moving the other network is not an option, you can change the Weave configuration instead, using the following procedure:

1. Edit the following file local to the ICM container:

   ```
   /ICM/etc/toHost/installWeave.sh
   ```

2. Find the line containing the string **weave launch**. If you're confident there is no danger of overlap between Weave and the existing network, you can force Weave to continue use the default range by adding the underscored text in the following:

```
sudo /usr/local/bin/weave launch --ipalloc-range 10.32.0.0/12 --password $2
```

You can also simply move Weave to another private network, as follows:

```
sudo /usr/local/bin/weave launch --ipalloc-range 172.30.0.0/16 --password $2
```

3. Save the file.
4. Reprovision the cluster.

4.9.6 Huge Pages

On certain architectures you may see an error similar to the following in the InterSystems IRIS messages log:

```
0 Automatically configuring buffers
1 Insufficient privileges to allocate Huge Pages; non-root instance requires CAP_IPC_LOCK capability
for Huge Pages.
2 Failed to allocate 1316MB shared memory using Huge Pages. Startup will retry with standard pages. If
 huge pages
  are needed for performance, check the OS settings and consider marking them as required with the
InterSystems IRIS
  'memlock' configuration parameter.
```

This can be remedied by providing the following option to the icm run command:

```
-options "--cap-add IPC_LOCK"
```

4.10 ICM Configuration Parameters

These tables describe the fields you can include in the configuration files (see Configuration, State and Log Files) to provide ICM with the information it needs to execute provisioning and deployment tasks and management commands.

- General Parameters
- Security-Related Parameters
- Provider-Specific Parameters
- Device Name Parameters
- Generated Parameters

4.10.1 General Parameters

Parameter	Meaning
LicenseDir	Location of InterSystems IRIS license keys staged in the ICM container and individually specified by the LicenseKey field (below); see InterSystems IRIS Licensing for ICM.
LicenseKey	License key for the InterSystems IRIS instance on one or more provisioned DM, AM, DS, or QS nodes, staged within the ICM container in the location specified by the LicenseDir field (above).
ISCPassword	Password that will be set for the _SYSTEM, Admin, and SuperUser accounts on the InterSystems IRIS instances on one or more provisioned nodes. Corresponding command-line option: **-iscPassword**.

Parameter	Meaning
DockerImage	Docker image to be used for icm run and icm pull commands. Must include the repository name. Corresponding command-line option: **-image**.
DockerRegistry	DNS name of the server hosting the Docker repository storing the image specified by DockerImage. If not included, ICM uses Docker's public registry located at registry-1.docker.io.
DockerUsername	Username to use for Docker login to the respository specified in DockerImage on the registry specified by DockerRegistry. Not required for public repositories. If not included and the repository specified by DockerImage is private, login fails.
DockerPassword	Password to use for Docker login, along with DockerUsername. Not required for public repositories. If this field and the repository specified by DockerImage is private. ICM prompts you (with masked input) for a password. (If the value of this field contains special characters such as **$**, **\|**, **(**, and **)**, they must be escaped with two **** characters; for example, the password **abc$def** must be specified as **abc\\\\$def**.)
DockerVersion	Version of Docker installed on provisioned nodes. Default is ce-18.03.1.ce. **Important:** The Docker images from InterSystems optionally deployed by ICM comply with the OCI support specification, and are supported on Enterprise Edition and Community Edition 18.03 and later. Docker Enterprise Edition only is supported for production environments. Not all combinations of platform and Docker version are supported by Docker; for detailed information from Docker on compatibility, see the Compatibility Matrix and About Docker CE.
DockerURL	URL of the Docker Enterprise Edition repository associated with your subscription or trial; when provided, triggers installation of Docker Enterprise Edition on provisioned nodes, instead of Docker Community Edition.
DockerStorageDriver	Determines the storage driver used by Docker. Values include overlay2 and devicemapper (the default). **Note:** • If DockerStorageDriver is set to overlay2, the FileSystem parameter must be set to xfs. • If DockerStorageDriver is set to overlay2, the DockerDeviceName parameter (see Device Name Parameters) must be set to null.
Home	Root of the home directory on a provisioned node. Default: /home.
Mirror	If True, InterSystems IRIS instances are deployed with mirroring enabled; see Mirrored Configuration Requirements. Default: False.
Spark	If True, and the **spark** image is specified by the DockerImage field, Spark is deployed with InterSystems IRIS in the **iris** container (see The icm run Command for more information). Default: false.

Parameter	Meaning
LoadBalancer	Set to True in definitions of node type AM, WS, VM, or CN for automatic provisioning of load balancer on providers AWS, GCP, and Azure. Default: false.
Namespace	Namespace to be created during deployment. The namespace specified is also set as the default namespace for the icm session and icm sql commands. The predefined namespace USER may be used in nonmirrored deployments only. For more information, see The Definitions File. Default: USER. Command-line option: **-namespace**.
ISCglobals *	Database cache allocation from system memory. See globals in the *Parameter File Reference* and Memory and Startup Settings in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*, Default: 0,0,0,0,0,0 (automatic allocation)
ISCroutines *	Routine cache allocation from system memory. See routines in the *Parameter File Reference* and Memory and Startup Settings in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*. Default: 0 (automatic allocation).
ISCgmheap *	Size of the generic memory heap (in KB). See gmheap in the *Parameter File Reference* and **gmheap** in the "Advanced Memory Settings" section of the *Additional Configuration Settings Reference*. . Default: 37568.
ISClocksiz *	Maximum size of shared memory for locks (in bytes). See locksiz in the *Parameter File Reference*. Default: 16777216.
ISCbbsiz *	Maximum memory per process (KB). See bbsiz in the *Parameter File Reference*. Default: 262144.
ISCmemlock *	Enable/disable locking shared memory or the text segment into memory. See memlock in the *Parameter File Reference*. Default: 0.
Overlay	Determines the Docker overlay network type; normally "weave", but may be set to "host" for development or debug purposes, or when deploying on a preexisting cluster. Default: weave (host when deploying on a preexisting cluster).
ISCAgentPort	Port used by InterSystems IRIS ISC Agent. Default: 2188.
JDBCGatewayPort *	Port used by InterSystems IRIS JDBC Gateway. Default: 62972.
SuperServerPort *	Port used by InterSystems IRIS Superserver. Default: 51773.
WebServerPort *	Port used by InterSystems IRIS Web Server (Management Portal). Default: 52773.
LicenseServerPort	Port used by InterSystems IRIS License Server. Default: 4002.
SparkMasterPort	Port used by Spark Master. Default: 7077.
SparkWorkerPort	Port used by Spark Worker. Default: 7000.
SparkMasterWebUIPort	Port used by Spark Master Web UI. Default: 8080.
SparkWorkerWebUIPort	Port used by Spark Worker Web UI. Default: 8081.
SparkRESTPort	Port used for Spark REST API. Default: 6066.

Parameter	Meaning
SparkDriverPort	Port used for Spark Driver. Default: 7001.
SparkBlocKManagerPort	Port used for Spark Block Manager. Default: 7005.
Count	Number of nodes to provision from a given entry in the definitions file. Default: 1.
Label	Name shared by all nodes in this deployment, for example ACME; cannot contain dashes.
Role	Role of the node or nodes to be provisioned by a given entry in the definitions file, for example DM or DS; see ICM Node Types.
Tag	Additional name used to differentiate between deployments, for example TEST; cannot contain dashes.
StartCount	Numbering start for a particular node definition in the definitions file. For example, if the DS node definition includes "StartCount": "3", the first DS node provisioned is named *Label*-DS-*Tag*-0002.
FileSystem	Type of file system to use for persistent volumes. Valid values are ext2, ext3, ext4, xfs, and btrfs. Default: xfs. **Note:** See the DockerStorageDriver parameter for restrictions.
ApplicationPath	Application path to create for definitions of type WS. Default: none.
AlternativeServers	Remote server selection algorithm for definitions of type WS. Valid values are LoadBalancing and FailOver. Default: LoadBalancing.
DataMountPoint	The location on the compute instance where the persistent volume described by DataDeviceName (see Device Name Parameters) will be mounted. Default: /irissys/data.
WIJMountPoint *	The location on the compute instance where the persistent volume described by WIJDeviceName (see Device Name Parameters) will be mounted. Default: /irissys/wij.
Journal1MountPoint *	The location on the compute instance where the persistent volume described by Journal1DeviceName (see Device Name Parameters) will be mounted. Default: /irissys/journal1j.
Journal2MountPoint *	The location on the compute instance where the persistent volume described by Journal2DeviceName (see Device Name Parameters) will be mounted. Default: /irissys/journal2j.
OSVolumeSize	Size (in GB) of the OS volume to create for deployments other than type PreExisting. Default: 32. **Note:** In some cases, this setting must be greater than or equal to a value specific to the OS image template or snapshot; for more information, see Creating a Virtual Machine from a Template in the Terraform documentation..

Parameter	Meaning
DockerVolumeSize	Size (in GB) of the block storage device used for the Docker thin pool for providers other than *PreExisting*. This volume corresponds to the DockerDeviceName parameter (see Device Name Parameters) . Default: 10.
DataVolumeSize	Size (in GB) of the persistent data volume to create for deployments other than type PreExisting. This volume corresponds to the DataDeviceName parameter (see Device Name Parameters) and will be mounted at DataMountPoint. Default: 10.
WIJVolumeSize	Size (in GB) of the persistent WIJ volume to create for deployments other than PreExisting. This volume corresponds to the WIJDeviceName parameter (see Device Name Parameters) and will be mounted at WIJMountPoint. Default: 10.
Journal1VolumeSize	Size (in GB) of the persistent Journal volume to create for deployments other than type PreExisting. This volume corresponds to the Journal1DeviceName parameter (see Device Name Parameters) and will be mounted at Journal1MountPoint. Default: 10.
Journal2VolumeSize	Size (in GB) of the alternate persistent Journal volume to create for deployments other than type PreExisting. This volume corresponds to the Journal2DeviceName parameter (see Device Name Parameters) and will be mounted at Journal2MountPoint. Default: 10.
SystemMode	String to be shown in the Management Portal masthead. Certain values (LIVE, TEST, FAILOVER, DEVELOPMENT) trigger additional changes in appearance. Default: blank.
Provider	Platform to provision infrastructure on; see Provisioning Platforms. Default: none.
Monitor	Deploy Weave Scope for basic monitoring by sepcifying the value scope. Default: none.
MonitorUsername	Username to use in authenticating to Weave Scope. Default: none.
MonitorPassword	Password to use in authenticating to Weave Scope. Default: none.
ProxyImage	Docker image used to provide authentication and HTTPS for Weave Scope monitoring.
ForwardPort	Port to be forwarded by a given load balancer (both 'from' and 'to'). Defaults: • AM: SuperServerPort • WS: 80 • VM/CN: (user provided)
ForwardProtocol	Protocol to be forwarded by a given load balancer. Defaults: • AM: tcp • WS: http • VM/CN: (user provided)

Parameter	Meaning
HealthCheckPort	Port used to verify health of instances in the target pool. Defaults: • AM: SuperServerPort • WS: 80 • VM/CN: (user provided)
HealthCheckProtocol	Protocol used to verify health of instances in the target pool. Defaults: • AM: tcp • WS: http • VM/CN: (user provided)
HealthCheckPath	Path used to verify health of instances in the target pool. Defaults: • AM: /csp/user/isc_status.cxw • WS: N/A (path not used for TCP health checks) • VM/CN: (user provided for HTTP health checks)

* The following parameters in the preceding table map directly to parameters in the iris.cpf file of the InterSystems IRIS instance on nodes of type DM, AM, DS, and QS:

ICM Name	iris.cpf Name
SuperServerPort	DefaultPort
WebServerPort	WebServerPort
JDBCGatewayPort	JDBCGatewayPort
WIJMountPoint	wijdir
Journal1MountPoint	CurrentDirectory
Journal2MountPoint	AlternateDirectory
ISCglobals	globals
ISCroutines	routines
ISCgmheap	gmheap
ISCbbsiz	bbsiz
ISClocksiz	locksiz
ISCmemlock	memlock

The LicenseServerPort field appears in the **[LicenseServers]** block of the iris.cpf file, bound to the name of the configured license server (see InterSystems IRIS Licensing for ICM).

4.10.2 Security-Related Parameters

The parameters in the following table are used to identify files and information required for ICM to communicate securely with the provisioned nodes and deployed containers.

- For information about using scripts provided with ICM to generate these files, see Obtain Security-Related Files in this document.
- For information about how ICM uses the security files you provide to communicate securely with provisioned nodes and services on them, see ICM Security.
- For general information about using the SSH protocol, see SSH PROTOCOL from SSH Communications Security.
- For information about using SSL/TLS with Docker, see Protect the Docker daemon socket in the Docker documentation.
- For general information about using SSL/TLS with InterSystems IRIS, see Using SSL/TLS with InterSystems IRIS and The InterSystems Public Key Infrastructure in the *Security Administration Guide*. For information about the contents of the file identified by the SSLConfig parameter, see Creating a Client Configuration in the same document.
- For information about the use of SSL/TLS to secure connections between mirror members, see Securing Mirror Communication with SSL/TLS Security in the *High Availability Guide*.

Parameter	Meaning

Parameter	Meaning
SSHUser	Nonroot account with sudo access used by ICM for access to provisioned nodes. Root of SSHUser's home directory can be specified using the `Home` field. Required value is provider-specific, as follows: • AWS — As per AMI specification (usually "ec2-user" for Red Hat Enterprise Linux instances) • vSphere — As per VM template • Azure — At user's discretion • GCP — At user's discretion
SSHPassword	Initial password for the user specified by SSHUser. Required for marketplace Docker images and deployments of type vSphere, Azure, and PreExisting. This is used only during provisiong, at the conclusion of which password logins are disabled.
SSHOnly	If True, ICM does not attempt SSH password logins during provisioning (providers PreExisting and vSphere only). Default: False.

SSHPublicKey	Public key of SSH public/private key pair; required for all deployments.
	For provider AWS, must be in SSH2 format, for example:
	`---- BEGIN SSH2 PUBLIC KEY ---`
	`AAAAB3NzaC1yc2EAAAABJQAAAQEAoa0`
	`---- BEGIN SSH2 PUBLIC KEY ---`
	For other providers, must be in OpenSSH format, for example:
	`ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEAoa0`

SSHPrivateKey	Private key of SSH public private key pair, required, in RSA format, for example:
	`-----BEGIN RSA PRIVATE KEY-----`
	`MIIEogIBAAKCAQEAoa0ex+JKzC2Nka1`
	`-----END RSA PRIVATE KEY-----`

TLSKeyDir	Directory containing TLS keys used to establish secure connections to Docker, InterSystems Web Gateway, JDBC, and mirrored InterSystems IRIS databases, as follows: • ca.pem • cert.pem • key.pem • keycert.pem • server-cert.pem • server-key.pem • keystore.p12 • truststore.jks • SSLConfig.properties

Parameter	Meaning
SSLConfig	Path to an SSL/TLS configuration file used to establish secure JDBC connections. Default: If this parameter is not provided, ICM looks for a configuration file in /*TLSKeyDir*/SSLConfig.Properties (see previous entry).

4.10.3 Provider-Specific Parameters

This tables in this section list parameters used by ICM that are specific to each provider, as follows:

- Selecting Machine Images
- Amazon Web Services (AWS) Parameters
- Google Cloud Platform (GCP) Parameters
- Microsoft Azure (Azure) Parameters
- VMware vSphere (vSphere) Parameters
- PreExisting Cluster (PreExisting) Parameters

Note: Some of the parameters listed are used with more than one provider.

4.10.3.1 Selecting Machine Images

Cloud providers operate data centers in various regions of the world, so one of the important things to customize for your deployment is the region in which your cluster will be deployed. Another choice is which virtual machine images to use for the computes nodes in your cluster. Although the sample configuration files define valid regions and machine images for all cloud providers, you will generally want to change the region to match your own location. Because machine images are often specific to a region, both must be selected.

At this release, ICM supports provisioning of and deployment on compute nodes running Red Hat Enterprise Linux, version 7.2 or later, so the machine images you select must run this operating system.

4.10.3.2 Amazon Web Services (AWS) Parameters

Parameter	Meaning

Credentials	Path to a file containing Amazon AWS credentials in the following format:
	`[default]aws_access_key_id = XXXXXXXXXXXXXXXXXX`
	`aws_secret_access_key = YYYYYYYYYYYYYYYYYYYY`
	Download from https://console.aws.amazon.com/iam/home?#/users.

SSHUser	Nonroot account with sudo access used by ICM for access to provisioned nodes (see Security-Related Parameters). Root of SSHUser's home directory can be specified using the Home field. Required value is determined by the selected AMI; for Red Hat Enterprise Linux images, the required value of SSHUser is usually **ec2-user.**
AMI	AMI to use for a node or nodes to be provisioned; see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html. Example: ami-a540a5e1.

Parameter	Meaning
Region	Region to use for a node or nodes to be provisioned; see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html. Example: us-west-1.
Zone	Availability zone to use for a node or nodes to be provisioned; see link in previous entry. Example: us-west-1c. For information on using this field with multiple zones, see Deploying Across Multiple Zones.
ZoneMap	When multiple zones are specified, specifies which nodes are deployed in which zones. Default: 0,1,2,...,255. For information on using this field with multiple zones, see Deploying Across Multiple Zones.
ElasticIP	Enables the Elastic IP feature to preserve IP address and domain name across compute node restart; for more information, see AWS Elastic IP Feature. Default: False.
InstanceType	Instance Type to use for a node or nodes to be provisioned; see https://aws.amazon.com/ec2/instance-types/. Example: m4.large.
VPCId	Existing Virtual Private Cloud (VPC) to be used in the deployment, instead of allocating a new one; the specified VPC is not deallocated during unprovision. If not specified, a new VPC is allocated for the deployment and deallocated during unprovision. **Note:** Internal parameter net_subnet_cidr must be provided if the VPC is not created in the default address space 10.0.0.0/16; for example, for a VPC in the range 172.17.0.0/16, you would need to specify net_subnet_cidr as 172.17.%d.0/24.
OSVolumeType	Determines maximum OSVolumeSize. See http://docs.aws.amazon.com/cli/latest/reference/ec2/create-volume.html. Default: standard.
OSVolumeIOPS	IOPS count for OSVolume. Must be nonzero for volumes of type iops. Default: 0.
DockerVolumeType	Determines maximum DockerVolumeSize (see OSVolumeType). Default: standard.
DockerVolumeIOPS	IOPS count for DockerVolume. Must be nonzero for volumes of type iops. Default: 0.
DataVolumeType	Determines maximum DataVolumeSize (see OSVolumeType). Default: standard.
DataVolumeIOPS	IOPS count for DataVolume. Must be nonzero for volumes of type iops. Default: 0.
WIJVolumeType	Determines maximum WIJVolumeSize (see OSVolumeType). Default: standard.
WIJVolumeIOPS	IOPS count for WIJVolume. Must be nonzero for volumes of type iops. Default: 0.
Journal1VolumeType	Determines maximum Journal1VolumeSize (see OSVolumeType). Default: standard.

Parameter	Meaning
Journal1VolumeIOPS	IOPS count for Journal1Volume. Must be nonzero for volumes of type iops. Default: 0.
Journal2VolumeType	Determines maximum Journal2VolumeSize (see OSVolumeType). Default: standard.
Journal2VolumeIOPS	IOPS count for Journal2Volume. Must be nonzero for volumes of type iops. Default: 0.

4.10.3.3 Google Cloud Platform (GCP) Parameters

Parameter	Meaning
Credentials	JSON file containing account credentials. Download from https://console.developers.google.com/
Project	Google project ID.
MachineType	Machine type resource to use for a node or nodes to be provisioned. See https://cloud.google.com/compute/docs/machine-types. Example: n1-standard-1.
Region	Region to use for a node or nodes to be provisioned; see https://cloud.google.com/compute/docs/regions-zones/regions-zones. Example: us-east1.
Zone	Zone in which to locate a node or nodes to be provisioned. Example: us-east1-b. For information on using this field with multiple zones, see Deploying Across Multiple Zones.
ZoneMap	When multiple zones are specified, specifies which nodes are deployed in which zones. Default: 0,1,2,...,255. For information on using this field with multiple zones, see Deploying Across Multiple Zones.
Image	The source image from which to create this disk. See https://cloud.google.com/compute/docs/images. Example: centos-cloud/centos-7-v20160803.
OSVolumeType	Determines disk type for the OS volume. See https://cloud.google.com/compute/docs/reference/beta/instances/attachDisk. Default: pd-standard.
DockerVolumeType	Determines disk type for the Docker block storage device (see OSVolumeType). Default: pd-standard.
DataVolumeType	Determines disk type for the persistent Data volume (see OSVolumeType). Default: pd-standard.
WIJVolumeType	Determines disk type for the persistent WIJ volume (see OSVolumeType). Default: pd-standard.
Journal1VolumeType	Determines disk type for the persistent Journal1 volume (see OSVolumeType). Default: pd-standard.
Journal2VolumeType	Determines disk type for the persistent Journal1 volume (see OSVolumeType). Default: pd-standard.

4.10.3.4 Microsoft Azure (Azure) Parameters

Parameter	Meaning
Size	The size of a node or nodes to be provisioned; see https://docs.microsoft.com/en-us/azure/virtual-machines/virtual-machines-windows-sizes. Example: Standard_DS1.
Location	Location in which to provision a node or nodes; see https://azure.microsoft.com/en-us/regions/. Example: Central US.
Zone	Zone in which to locate a node or nodes to be provisioned. Possible values are 1, 2, and 3.
SubscriptionId	Credentials which uniquely identify the Microsoft Azure subscription.
ClientId	Azure application identifier.
ClientSecret	Provides access to an Azure application.
TenantId	Azure Active Directory tenant identifier.
PublisherName	Entity providing a given Azure image. Example: OpenLogic.
Offer	Operating system of a given Azure image. Example: Centos.
Sku	Major version of the operating system of a given Azure image. Example: 7.2.
Version	Build version of a given Azure image. Example: 7.2.20170105.
AccountTier	Account tier, either HDD (Standard) or SSD (Premium).
AccountReplicationType	Account storage type: locally-redundant storage (*LRS*), geo-redundant storage (*GRS*), zone-redundant storage (*ZRS*), or read access geo-redundant storage (*RAGRS*).
ResourceGroupName	Existing Resource Group to be used in the deployment, instead of allocating a new one; the specified group is not deallocated during unprovision. If not specified, a new Resource Group is allocated for the deployment and deallocated during unprovision.
VirtualNetworkName	Existing Virtual Network to be used in the deployment, instead of allocating a new one; the specified network is not deallocated during unprovision. If not specified, a new Virtual Network is allocated for the deployment and deallocated during unprovision. **Note:** Internal parameter net_subnet_cidr must be provided if the network is not created in the default address space 10.0.%d.0/24.
SubnetId	Existing Subnet to be used in the deployment, instead of allocating a new one; the specified subnet is not deallocated during unprovision. If not specified, a new Subnet is allocated for the deployment and deallocated during unprovision. Value is an Azure URI of the form: `/subscriptions/<subscription>/resourceGroups/<resource_group>/providers /Microsoft.Network/virtualNetworks/<virtual_network>/subnets/<subnet_name>`

Parameter	Meaning
UseMSI	When True, authenticates using a Managed Service Identity in place of ClientId and ClientSecret. Requires that ICM be run from a machine in Azure. Default: false
CustomImage	Image to be used to create the OS disk, in place of the marketplace image described by the PublisherName, Offer, Sku, and Version fields. Value is an Azure URI of the form: `/subscriptions/<subscription>/resourceGroups/<resource_group>/providers /Microsoft.Compute/images/<image_name>`

4.10.3.5 VMware vSphere (vSphere) Parameters

Parameter	Meaning
Server	Name of the vCenter server. Example: tbdvcenter.iscinternal.com.
Datacenter	Name of the datacenter.
VsphereUser	Username for vSphere operations.
VspherePassword	Password for vSphere operations.
VCPU	Number of CPUs in a node or nodes to be provisioned. Example: 2.
Memory	Amount of memory (in MB) in a node or nodes to be provisioned. Example: 4096.
DatastoreCluster	Collection of datastores where virtual machine files will be stored. Example: DatastoreCluster1.
DNSServers	List of DNS servers for the virtual network. Example: 172.16.96.1,172.17.15.53
DNSSuffixes	List of name resolution suffixes for the virtual network adapter. Example: iscinternal.com
Domain	FQDN for a node to be provisioned. Example: iscinternal.com
NetworkInterface	Label to assign to a network interface. Example: VM Network
Template	Virtual machine master copy. Example: centos-7
GuestID	Guest ID for the operating system type. See Enum - VirtualMachineGuestOsIdentifier on the VMware support website. Default: centos64Guest.
WaitForGuestNetTimeout	Time (in minutes) to wait for an available IP address on a virtual machine. Default: 5.
ShutdownWaitTimeout	Time (in minutes) to wait for graceful guest shutdown when making necessary updates to a virtual machine. Default: 3.
MigrateWaitTimeout	Time (in minutes) to wait for virtual machine migration to complete. Default: 10.
CloneTimeout	Time (in minutes) to wait for virtual machine cloning to complete. Default: 30.

Parameter	Meaning
CustomizeTimeout	Time (in minutes) that Terraform waits for customization to complete. Default: 10.
DiskPolicy	Disk provisioning policy for the deployment (see About Virtual Disk Provisioning Policies in the VMware documentation). Values are: • thin — Thin Provision • lazy — Thick Provision Lazy Zeroed • eagerZeroedThick — Thick Provision Eager Zeroed Default: lazy.
ResourcePool	Name of a vSphere resource pool. Example: ResourcePool1.
SDRSEnabled	If specified, determines whether Storage DRS is enabled for a virtual machine; otherwise, use current datastore cluster settings. Default: Current datastore cluster settings.
SDRSAutomationLevel	If specified, determines Storage DRS automation level for a virtual machine; otherwise, use current datastore cluster settings. V;ues are automated or manual. Default: Current datastore cluster settings.
SDRSIntraVMAffinity	If provided, determines Intra-VM affinity setting for a virtual machine; otherwise, use current datastore cluster settings. Values include: • True — All disks for this virtual machine will be kept on the same datastore. • False — Storage DRS may locate individual disks on different datastores if it helps satisfy cluster requirements. Default: Current datastore cluster settings.
SCSIControllerCount	Number of SCSI controllers for a given compute instance; must be between 1 and 4. The OS volume is always be placed on the first SCSI controller. vSphere may not be able to create more SCSI controllers than were present in the template specified by the Template field. Default: 1
DockerVolumeSCSICon-troller	SCSI controller on which to place the Docker volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1
DataVolumeSCSICon-troller	SCSI controller on which to place the Data volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1
WIJVolumeSCSIController	SCSI controller on which to place the WIJ volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1

Parameter	Meaning
Journal1VolumeSCSICon-troller	SCSI controller on which to place the Journal1 volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1
Journal2VolumeSCSICon-troller	SCSI controller on which to place the Journal2 volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1

Note: The requirements for the VMware vSphere template are similar to those described in Compute Node Requirements for preexisting clusters (for example, passwordless sudo access).

To address the needs of the many users who rely on VMware vSphere, it is supported by this release of ICM. Depending on your particular vSphere configuration and underlying hardware platform, the use of ICM to provision virtual machines may entail additional extensions and adjustments not covered in this guide, especially for larger and more complex deployments, and may not be suitable for production use. Full support is expected in a later release.

4.10.3.6 PreExisting Cluster (PreExisting) Parameters

Parameter	Meaning
IPAddress	This is a required field (in the definitions file) for provider PreExisting and is a generated field for all other providers.
DNSName	FQDN of the compute instance, or its IP Address if unavailable. Deployments of type PreExisting may populate this field (in the definitions file) to provide names for display by the **icm inventory** command. This is a generated field for all other providers.

4.10.4 Device Name Parameters

The parameters in the following table specify the devices (under /dev) on which persistent volumes appear (see Storage Volumes Mounted by ICM). Defaults are available for all providers other than PreExisting, but these values are highly platform and OS-specific and may need to be overridden in your defaults.json file. For PreExisting deployments, see Storage Volumes in the "Deploying on a Preexisting Cluster" appendix.

Parameter	AWS	GCP	vSphere	Azure
DockerDeviceName	xvdb	sdb	sdb	sdc
DataDeviceName	xvdc	sdc	sdc	sdd
WIJDeviceName	xvdd	sdd	sdd	sde
Journal1DeviceName	xvde	sde	sde	sdf
Journal2DeviceName	xvdf	sdf	sdf	sdg

Note: For restrictions on DockerDeviceName, see the DockerStorageDriver parameter (see General Parameters).

4.10.5 Generated Parameters

These parameters are generated by ICM during provisioning, configuration, and deployment. They should generally be treated as read-only and are included here for information purposes only.

Parameter	Meaning
Member	In initial configuration of a mirrored pair, set to primary, backup, or async. (Actual role of each failover member is determined by mirror operations.)
MachineName	Generated from *Label-Role-Tag-####*.
WeaveArgs	Generated by executing **weave dns-args** on the compute instance.
WeavePeers	List of IP addresses of every compute instance except the current on.
WeavePassword	Password used to encrypt traffic over Weave Net; disable encryption by setting to the literal "null" in the defaults.json file.
StateDir	Location on the ICM client where temporary, state, and log files will be written. Default: ICM-nnnnnnnnn. Command-line option: **-stateDir**.
DefinitionIndex	Assigns an index to each object in the definitions.json file; this is used to uniquely number load balancer instances (which would otherwise have the same names).
TargetRole	The role associated with the resources being managed by a given load balancer.
MirrorSetName	Name assigned to a failover mirror.
InstanceCount	Total number of instances in this deployment.

A
Containerless Deployment

If you want to use ICM to provision cloud infrastructure and deploy noncontainerized InterSystems IRIS instances on that infrastructure, or to install InterSystems IRIS on a PreExisting cluster, you can do so using containerless mode.

In essence, containerless mode replaces the containerized deployment of InterSystems IRIS by ICM with direct installation from traditional kits, while retaining all the other steps in the ICM provisioning and deployment process. This is accomplished by adding two commands to ICM and adapting several others.

In containerless mode, Docker is not installed on the provisioned nodes and the **icm run** command cannot be used to deploy containers on those nodes.

Note: Sharded configurations cannot be deployed on GCP using containerless mode. The Web Gateway cannot be deployed on Ubuntu or SUSE nodes using containerless mode.

- Enabling Containerless Mode
- Installing InterSystems IRIS
- Uninstalling InterSystems IRIS
- Additional Containerless Mode Commands

A.1 Enabling Containerless Mode

Enable containerless mode by adding the Containerless field to the defaults.json file with a value of True, for example:

```
{
    "Containerless": "True",
    "Provider": "AWS",
    "Label": "ACME",
    "Tag": "TEST"
    "LicenseDir": "/Samples/license/",
    "Credentials": "/Samples/AWS/sample.credentials",
    ...
}
```

A.2 Installing InterSystems IRIS

To install InterSystems IRIS on your provisioned nodes using the installation kit you have selected, use the **icm install** command, which does not exist in container mode. The kit is identified by the KitURL field, which specifies the path to the installation kit and can be added to either defaults.json or definitions.json. The specified kit must be all of the following:

- Accessible by the node on which InterSystems IRIS is to be installed (though not necessarily by the ICM container itself)
- A 64-bit Linux kit
- A gzipped tar file

For example, in the definitions file:

```
[
    {
        "Role": "DM",
        "Count": "1",
        "DataVolumeSize": "50",
        "InstanceType": "m4.xlarge",
        "KitURL": "http://kits.acme.com/iris/2018.2.0/unix/IRIS-2018.2.0.792.0-lnxrhx64.tar.gz"
    },
    {
        "Role": "AM",
        "Count": "2",
        "StartCount": "2",
        "LoadBalancer": "true",
        "KitURL": "http://kits.acme.com/iris/2018.2.0/unix/IRIS-2018.2.0.792.0-lnxrhx64.tar.gz"
    }
]
```

In the defaults file:

```
{
    "Containerless": "True",
    "KitURL": "http://kits.acme.com/iris/2018.2.0/unix/IRIS-2018.2.0.792.0-lnxrhx64.tar.gz"
    "Provider": "AWS",
    "Label": "ACME",
    "Tag": "TEST"
    "LicenseDir": "/Samples/license/",
    "Credentials": "/Samples/AWS/sample.credentials",
    ...
}
```

Note: The KitURL can be a reference to a local file copied to the provisioned nodes, which may be convenient under some circumstances. For example, you can include this KitURL in the defaults file:

```
"KitURL": "file://tmp/IRIS-2018.2.0.792.0-lnxrhx64.tar.gz"
```

and use the **icm scp** command to copy the kit to the provisioned nodes before executing the **icm install** command, for example:

```
icm scp -localFile IRIS-2018.2.0.792.0-lnxrhx64.tar.gz -remoteFile /tmp
```

When you execute the **icm install** command, ICM installs InterSystems IRIS from the specified kit on each applicable node, resulting in output like the following:

```
Downloading kit on ACME-DM-TEST-0001...
Downloading kit on ACME-AM-TEST-0002...
Downloading kit on ACME-AM-TEST-0003...
...downloaded kit on ACME-AM-TEST-0002
...downloaded kit on ACME-AM-TEST-0003
...downloaded kit on ACME-DM-TEST-0001
Installing kit on ACME-AM-TEST-0003...
Installing kit on ACME-DM-TEST-0001...
Installing kit on ACME-AM-TEST-0002...
...installed kit on ACME-AM-TEST-0002
...installed kit on ACME-DM-TEST-0001
...installed kit on ACME-AM-TEST-0003
Starting InterSystems IRIS on ACME-DM-TEST-0001...
Starting InterSystems IRIS on ACME-AM-TEST-0002...
Starting InterSystems IRIS on ACME-AM-TEST-0003...
...started InterSystems IRIS on ACME-AM-TEST-0002
...started InterSystems IRIS on ACME-AM-TEST-0003
...started InterSystems IRIS on ACME-DM-TEST-0001
Management Portal available at: http://172.16.110.14:52773/csp/sys/UtilHome.csp
```

A.3 Uninstalling InterSystems IRIS

The **icm uninstall** command, which does not exist in container mode, is used in containerless mode to stop and uninstall all InterSystems IRIS instances in the deployment (without options). You can use the **-role** and **-machine** options, as usual, to limit the command to a specific role or node. For example,

```
icm uninstall
```

uninstalls InterSystems IRIS on all nodes in the deployment, while

```
icm uninstall -role AM
```

uninstalls InterSystems IRIS on the AM nodes only.

A.4 Additional Containerless Mode Commands

Several container mode commands work in the same way, or an analagous way, in containerless mode, including use of the **-machine** and **-role** options, as follows:

- **icm ssh**, **icm scp**, **icm session**, **icm sql**

  The behavior of these commands is identical in container mode and containerless mode.

- **icm ps**

  The columns included in **icm ps** output in containerless mode are shown in the following example:

  ```
  # icm ps -json
  Machine            IP Address     Instance   Kit             Status   Health
  -------            ----------     --------   ---             ------   ------
  ACME-DM-TEST-0001  54.67.2.117    IRIS       2018.2.0.792.0  running  ok
  ACME-AM-TEST-0002  54.153.96.236  IRIS       2018.2.0.792.0  running  ok
  ACME-AM-TEST-0003  54.103.9.388   IRIS       2018.2.0.792.0  running  ok
  ```

  The **Instance** field provides the name of each instance (in a container provided by InterSystems, this is always `IRIS`) and the **Kit** field the kit from which it was installed. Values for **Status** include **running**, **down**, and **sign-on inhibited**; values for **Health** include **ok**, **warn**, and **alert**.

- **icm stop**, **icm start**

The **icm stop** and **icm start** commands execute the **iris stop** and **iris start** commands (see Controlling InterSystems IRIS Instances in the “Using Multiple Instances of InterSystems IRIS” chapter of the *System Administration Guide*) on all InterSystems IRIS instances or the specified instance(s).

- **icm upgrade**

  In containerless mode, **icm upgrade** does the following:

  – Downloads the InterSystems IRIS kit specified by KitURL.

  – Stops the current InterSystems IRIS instance using **iris stop**.

  – Uninstalls the current InterSystems IRIS instance.

  – Installs the InterSystems IRIS kit specified by KitURL.

  – Starts the newly-installed InterSystems IRIS instance using **iris start**.

  The following shows output from the **icm upgrade** command in containerless mode:

```
# icm ps
Machine            IP Address     Instance   Kit             Status   Health
-------            ----------     --------   ---             ------   ------
ACME-DM-TEST-0001  54.67.2.117    IRIS       2018.2.0.792.0  running  ok
ACME-AM-TEST-0002  54.153.96.236  IRIS       2018.2.0.792.0  running  ok
ACME-AM-TEST-0003  54.103.9.388   IRIS       2018.2.0.792.0  running  ok

# icm upgrade
Downloading kit on ACME-DM-TEST-0001...
Downloading kit on ACME-AM-TEST-0002...
Downloading kit on ACME-AM-TEST-0003...
...downloaded kit on ACME-AM-TEST-0002
...downloaded kit on ACME-AM-TEST-0003
...downloaded kit on ACME-DM-TEST-0001
Stopping InterSystems IRIS on ACME-DM-TEST-0001...
Stopping InterSystems IRIS on ACME-AM-TEST-0003...
Stopping InterSystems IRIS on ACME-AM-TEST-0002...
...stopped InterSystems IRIS on ACME-DM-TEST-0001
...stopped InterSystems IRIS on ACME-AM-TEST-0002
...stopped InterSystems IRIS on ACME-AM-TEST-0003
Uninstalling InterSystems IRIS on ACME-AM-TEST-0003...
Uninstalling InterSystems IRIS on ACME-DM-TEST-0001...
Uninstalling InterSystems IRIS on ACME-AM-TEST-0002...
...uninstalled InterSystems IRIS on ACME-DM-TEST-0001
...uninstalled InterSystems IRIS on ACME-AM-TEST-0002
...uninstalled InterSystems IRIS on ACME-AM-TEST-0003
Installing kit on ACME-AM-TEST-0002...
Installing kit on ACME-DM-TEST-0001...
Installing kit on ACME-AM-TEST-0003...
...installed kit on ACME-AM-TEST-0002
...installed kit on ACME-DM-TEST-0001
...installed kit on ACME-AM-TEST-0003
Starting InterSystems IRIS on ACME-DM-TEST-0001...
Starting InterSystems IRIS on ACME-AM-TEST-0002...
Starting InterSystems IRIS on ACME-AM-TEST-0003...
...started InterSystems IRIS on ACME-AM-TEST-0002
...started InterSystems IRIS on ACME-AM-TEST-0003
...started InterSystems IRIS on ACME-DM-TEST-0001

# icm ps
Machine            IP Address     Instance   Kit             Status   Health
-------            ----------     --------   ---             ------   ------
ACME-DM-TEST-0001  54.67.2.117    IRIS       2018.2.1.417.0  running  ok
ACME-AM-TEST-0002  54.153.96.236  IRIS       2018.2.1.417.0  running  ok
ACME-AM-TEST-0003  54.103.9.388   IRIS       2018.2.1.417.0  running  ok
```

B
Sharing ICM Deployments

ICM deployment results in the generation of several state files. Without access to those state files or the ICM container from which the cluster was deployed, it is difficult for anyone to manage or monitor the deployment (including the original deployer, should those files be lost).

This section describes which state files are required to share a deployment, methods of accessing them from outside the container, and how to persist those files so an ICM-driven deployment can be shared with other users or accessed from another location.

- State Files
- Maintaining Immutability
- Persisting State Files

B.1 State Files

The state files are read from and written to the current working directory, though all of them can be overridden to use a custom name and location. Input files are as follows:

- defaults.json — Override with **-defaults** *filepath*
- definitions.json — Override with **-definitions** *filepath*

Any security keys, InterSystems IRIS™ licenses, or other files referenced from within these configuration files should be considered input as well.

Output files are as follows:

- instances.json — Override with **-instances** *filepath*
- ICM-*GUID*/ — Override with **-stateDir** *path*

The layout of the files under ICM-*GUID*/ is as follows:

```
definition 0/
definition 1/
...
definition N/
```

Under each definition directory are the following files:

- terraform.tfvars — Terraform inputs

- terraform.tfstate — Terraform state

A variety of log files, temporary files, and other files appear in this hierarchy as well, but they are not required for sharing a deployment.

Note: For provider PreExisting, no Terraform files are generated.

B.2 Maintaining Immutability

InterSystems recommends that you avoid generating state files local to the ICM container, for the following reasons:

- Immutability is violated.
- Data can be lost if container removed/updated/replaced.
- Ability to edit configuration files within the ICM container is limited.
- Tedious and error-prone copying of state files out of the container is required.

A better practice is to mount a directory from the host within the ICM container to use as your working directory; that way all changes within the container are always available on the host. This can be accomplished using the Docker **--volume** option when the ICM container is first created, as follows:

```
$ docker run it -cap-add SYS_TIME --volume <host_path>:<container_path> <image>
```

Overall, you would take these steps:

1. Stage input files on the host in *host_path*.
2. Create, start, and attach to ICM container.
3. Navigate to *container_path*.
4. Issue ICM commands.
5. Exit or detach from ICM container.

The state files (both input and output) are then present in *host_path*. See the sample script in Launch ICM for an example of this approach.

B.3 Persisting State Files

Methods of preserving and sharing state files with others include:

- Make a tar/gzip

  The resulting archive can be emailed, put on an FTP site, a USB stick, and so on.
- Make backups to a location from which others can restore

  Register the path to the state files on the host with a backup service.
- Mount a disk volume accessible by others in your organization

  The path to the state files could be a Samba mount, for example.
- Specify a disk location backed up to the cloud

You might use services such as Dropbox, Google Drive, OneDrive, and so on.

- Store in a document database

  This could be cloud-based or on-premises.

The advantage of the latter three methods is that they allow others to modify the deployment. Note however that ICM does not support simultaneous operations issued from more than one ICM container at a time, so a policy ensuring exclusive read-write access would need to be enforced.

C
Scripting with ICM

This appendix describes how to issue a series of ICM commands from a script and how to identify and coordinate containers and services across a deployment.

- ICM Exit Status
- ICM Logging
- Remote Script Invocation
- Using JSON Mode

C.1 ICM Exit Status

ICM sets the UNIX exit status after each command, providing a simple way to determine whether a given ICM command succeeded. The following examples examine the special variable **$?** after each command:

```
# icm inventory
Machine            IP Address     DNS Name
-------             ---------      -------
ACME-DM-TEST-0001  54.191.233.2   ec2-54-191-233-2.amazonaws.com
ACME-DS-TEST-0002  54.202.223.57  ec2-54-202-223-57.amazonaws.com
ACME-DS-TEST-0003  54.202.223.58  ec2-54-202-223-58.amazonaws.com
# echo $?
0

# icm publish
Unrecognized goal: 'publish' (try "icm -help")
# echo $?
1

# icm ps -role QM
Unrecognized role 'QM'
# echo $?
1

# icm session
Error: Interactive commands cannot match more than one instance
# echo $?
1
```

C.2 ICM Logging

ICM logs its output to a file (default icm.log) and to the console. Whereas all output is sent to the log file, its console output can be captured and split into stdout and stderr. The following example completes without error:

```
# icm inventory > std.out 2> std.err
# cat std.out
Machine            IP Address     DNS Name
-------             ---------      -------
ACME-DM-TEST-0001  54.191.233.2   ec2-54-191-233-2.amazonaws.com
ACME-DS-TEST-0002  54.202.223.57  ec2-54-202-223-57.amazonaws.com
ACME-DS-TEST-0003  54.202.223.58  ec2-54-202-223-58.amazonaws.com
# cat std.err
```

The following example contains error output:

```
# icm publish > std.out 2> std.err
# cat std.out
# cat std.err
Unrecognized goal: 'publish' (try "icm -help")
```

C.3 Remote Script Invocation

Commands can be used in combination to copy scripts to a host or container and remotely invoke them. The following example copies an exported InterSystems IRIS™ routine into an InterSystems IRIS cluster, then compiles and runs it:

```
# icm scp -localPath Routine1.xml -remotePath /tmp/
# icm session -command 'Do ##class(%SYSTEM.OBJ).Load("/tmp/Routine1.xml", "c-d")'
# icm session -command 'Do ^Routine1'
```

This example copies a shell script to the host, changes its permissions, and executes it:

```
# icm scp -localPath script1.sh -remotePath /tmp/
# icm ssh -command 'sudo chmod a+x /tmp/script1.sh'
# icm ssh -command '/tmp/script1.sh abc 123'
```

This example does the same thing, but within a custom or third-party container:

```
# icm cp -localPath script2.sh -remotePath /tmp/ -container gracie
# icm exec -command 'chmod a+x /tmp/script2.sh' -container gracie
# icm exec -command '/tmp/script2.sh abc 123' -container gracie
```

C.4 Using JSON Mode

Your script may need to gather information from ICM about the state of the cluster. Examples would be:

- What is the IP address of the InterSystems IRIS Shard Master Data Server?
- What is the status of my custom/third-party container?

Parsing the human-readable output of ICM is difficult and prone to breakage. A more reliable solution is to have ICM generate its output in JSON format using the **json** option. The output is written to a file named response.json in the current working directory.

- Normal Output
- Abnormal Output

C.4.1 Normal Output

Most ICM commands do not result in any output upon success, in which case the exit value will be 0, no output will be written to stderr, and the JSON will be the empty array:

```
# icm exec -command "ls /" -json
# cat response.json
[]
```

ICM commands that produce useful output on success are detailed in the following. Note that the order of fields is not guaranteed.

C.4.1.1 icm provision

The format of the **icm provision** output is an object containing name-value pairs which describe the input (that is, configuration) and output (that is, state) files used during provisioning. The names, which match their corresponding command-line argument, are defaults, definitions, instances, and stateDir, as shown in this example:

```
# icm provision -json

Machine              IP Address       DNS Name
-------              ----------       --------
ACME-DM-TEST-0001    52.53.180.221    ec2-52-53-180-221.amazonaws.com
ACME-QS-TEST-0002    52.53.255.107    ec2-52-53-255-107.amazonaws.com
ACME-DS-TEST-0004    54.67.91.138     ec2-54-67-91-138.amazonaws.com
ACME-DS-TEST-0003    54.67.88.221     ec2-54-67-88-221.amazonaws.com
To destroy: icm unprovision -stateDir /Samples/AWS/ICM-3078941 [-cleanUp] [-force]

# cat response.json
{
  "defaults" : "defaults.json",
  "definitions" : "definitions.json",
  "instances" : "instances.json",
  "stateDir" : "/Samples/AWS/ICM-3078941/"
}
```

C.4.1.2 icm inventory

The format of the icm inventory command is an array whose elements correspond to each provisioned instance; each element in turn contains a list of the name-value pairs MachineName, Role, IPAddress, and DNSName, as shown in the following:

```
# icm inventory -json
Machine            IP Address     DNS Name
-------            ----------     --------
ACME-DM-TEST-0001  54.191.233.2   ec2-54-191-233-2.amazonaws.com
ACME-DS-TEST-0002  54.202.223.57  ec2-54-202-223-57.amazonaws.com
ACME-DS-TEST-0003  54.202.223.58  ec2-54-202-223-58.amazonaws.com

# cat response.json
[
  {
    "MachineName":"ACME-DM-TEST-0001",
    "Role":"DM",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
  },
  {
    "MachineName ":"ACME-DS-TEST-0002",
    "Role":"DS",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
  },
  {
    "MachineName":"ACME-DS-TEST-0002",
    "Role":"DS",
    "IPAddress":"54202.223.58",
    "DNSName":"54_202_223_58.amazonaws.com"
  }
]
```

C.4.1.3 icm ps

In container mode, the output of the icm ps command is an array whose elements correspond to each container; each element in turn is a list of the name-value pairs MachineName, Role, IPAddress, DNSName, Container, DockerImage, Status, and MirrorStatus (if applicable):

```
# icm ps -container iris -json
Machine            IP Address     Container  Status  Health  Image
-------            ----------     ---------  ------  -----   -----
```

```
ACME-DM-TEST-0001  54.191.233.2   iris       Up      healthy isc/iris:stable
ACME-DS-TEST-0002  54.202.223.57  iris       Up      healthy isc/iris:stable
ACME-DS-TEST-0003  54.202.223.58  iris       Up      healthy isc/iris:stable
# cat response.json
[
  {
    "MachineName":"ACME-DM-TEST-0001",
    "Role":"DM",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:stable",
    "Status":"Up",
    "Health":"healthy"
  },
  {
    "MachineName ":"ACME-DS-TEST-0002",
    "Role":"DS",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:stable",
    "Status":"Up"",
    "Health":"healthy"
  },
  {
    "MachineName ":"ACME-DS-TEST-0003",
    "Role":"DS",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:stable",
    "Status":"Up",
    "Health":"healthy"
  },
]
```

The icm ps output fields in containerless mode are MachineName, Role, IPAddress, DNSName, ISCInstance (always **IRIS** in images provided by InterSystems), Kit, Status, and MirrorStatus (if applicable):

```
# icm ps -json
Machine             IP Address      Instance   Kit              Status   Health
-------             ----------      --------   ---              ------   ------
ACME-DM-TEST-0001   54.67.2.117     IRIS       2017.3.0.392.0   running  ok
ACME-DS-TEST-0002   54.153.96.236   IRIS       2017.3.0.392.0   running  ok
ACME-DS-TEST-0002   54.153.90.66    IRIS       2017.3.0.392.0   running  ok

# cat response.json
[
  {
    "MachineName":"ACME-DM-TEST-0001",
    "Role":"DM",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running",
    "Health":"ok"
  },
  {
    "MachineName ":"ACME-DS-TEST-0002",
    "Role":"DS",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running"",
    "Health":"ok"
  },
  {
    "MachineName ":"ACME-DS-TEST-0003",
    "Role":"DS",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running"",
    "Health":"ok"
  }
]
```

C.4.2 Abnormal Output

When an error occurs, the format of the JSON output depends on whether the error occurred local to the ICM application or was from a target application on the host or instance.

C.4.2.1 Local Errors

When an ICM command results in an error, the JSON will contain an object and a name-value pair describing the error, as follows:

```
# icm ps -role QM -json
Unrecognized role 'QM'

# cat response.json
{
    "error": "Unrecognized role 'QM'"
}
```

C.4.2.2 Remote Errors

A remote error is considered to have occurred when one or more of the following is true:

- Non-zero exit status
- Output to stderr

When a remote error occurs, the JSON will be an array of objects containing name-value pairs; the name corresponds to that of the target machine, and the value is another object containing a list of name-value pairs including one or more of the following:

- **error**: A description of the problem that occurred; most of the text of an exception
- **file**: A file containing more detail about the problem
- **exitValue**: The (non-zero) exit value of an underlying process

Here is an example:

```
# icm ssh -command "ls file.txt" -json
Executing command 'ls file.txt' on host ACME-DM-TEST-0001...
ls: cannot access file.txt: No such file or directory
Error: See tmp/DM-TEST/DM-TEST-0001/ssh.err
Errors occurred during execution; aborting operation

# cat response.json
[
  {
    "ACME-DM-TEST-0001": {
      "file": "tmp/DM-TEST/DM-TEST-0001/ssh.err"
    }
  }
]

# cat tmp/DM-TEST/DM-TEST-0001/ssh.err
ls: cannot access file.txt: No such file or directory
```

D
Using ICM with Custom and Third-Party Containers

This appendix describes using ICM to deploy customer and third-party containers. Instructions assume that your Docker image resides in a repository accessible by ICM. For information on how to configure your container to communicate with other containers and services (including InterSystems IRIS™), see Scripting with ICM.

- Container Naming
- Overriding Default Commands
- Using Docker Options

D.1 Container Naming

Each container running on a given host must have a unique name. When deploying a container using icm run, the container can be named using the **-container** option:

```
# icm run -container gracie -image docker/whalesay
```

You can see the name reflected in the output of icm ps:

```
# icm ps
Machine            IP Address    Container   Status      Health   Image
-------             ---------     --------    -----      ------   ----
ACME-DM-TEST-0001   172.16.110.9  gracie      Restarting           docker/whalesay
```

Note: If the **-container** option is not provided, the default container name **iris** is used. Both **iris** and **spark** are reserved and should only be used for containers derived from InterSystems IRIS and Apache Spark Docker images provided by InterSystems.

D.2 Overriding Default Commands

If you want to override a container's default command, you can do so with -*command*. For example, suppose the docker/whalesay image runs command /bin/bash by default:

```
# icm docker -command "ps -a"

CONTAINER ID  IMAGE             COMMAND      CREATED     STATUS      NAMES
17f4ece54c2f  docker/whalesay  "/bin/bash"  4 days ago  Restarting  gracie
```

To have the container run a different command, such as **pwd**, you could deploy it as follows:

```
# icm run -container gracie -image docker/whalesay command pwd
```

You can verify that the command succeeded by examining the Docker logs:

```
# icm docker -command "logs gracie"
/cowsay
```

D.3 Using Docker Options

Your container may require Docker options or overrides not explicitly provided by ICM; these can be included using the **-options** option. This section provides examples a few of the more common use cases. For complete information about Docker options see https://docs.docker.com/engine/reference/run/.

- Restarting
- Privileges
- Environment Variables
- Mount Volumes
- Ports

D.3.1 Restarting

By default, ICM deploys containers with the option **--restart unless-stopped**. This means that if the container crosses an execution boundary for any reason other than an **icm stop** command (container exit, Docker restart, and so on), Docker keeps attempting to run it. In certain cases however, we want the container to run once and remain terminated. In this case, we can suppress restart as follows:

```
# icm run -container gracie -image docker/whalesay -options "--restart no"
# icm ps
Machine            IP Address    Container   Status      Health   Image
-------            ---------     --------    -----       ------   -----
ACME-DM-TEST-0001   172.16.110.9  gracie      Exited (0)           docker/whalesay
```

D.3.2 Privileges

Some containers require additional privileges to run, or you may want to remove default privileges. Examples:

```
# icm run -container sensors -image hello-world -options "--privileged"
# icm run -container fred -image hello-world -options "--cap-add SYS_TIME"
# icm run -container fred -image hello-world -options "--cap-drop MKNOD"
```

D.3.3 Environment Variables

Environment variables can be passed to your container using the Docker option **--env**. These variables are be set within your container in a manner similar to the bash **export** command:

```
# icm run container fred image hello-world options "--env TERM=vt100"
```

D.3.4 Mount Volumes

If your container needs to access files on the host machine, a mount point can be created within your container using the Docker **--volume** option. For example:

```
# icm run container fred image hello-world options "--volume /dev2:/dev2"
```

This makes the contents of directory /dev2 on the host available at mount point /dev2 within the container:

```
# icm ssh -command "touch /dev2/example.txt"  // on the host
# icm exec -command "ls /dev2"                // in the container
example.txt
```

D.3.5 Ports

Ports within your container can be mapped to the host using the Docker option **--publish**:

```
# icm run -container fred -image hello-world -options "--publish 80:8080"
# icm run -container fred -image hello-world -options "--publish-all"
```

You must open the corresponding port on the host if you wish to access the port from outside. This can be achieved in a number of ways, including:

- By editing the Terraform template file infrastructure.tf files directly.
- By issuing commands to the host using the icm ssh command.
- By modifying the security settings in the console of the cloud provider.

You also have to ensure that you are not colliding with a port mapped to another container or service on the same host. Finally, keep in mind that **--publish** has no effect on containers when the overlay network is of type **host**.

The following example modifies the Terraform template for AWS to allow incoming TCP communication over port 563 (NNTP over SSL/TLS):

- File: /ICM/etc/Terraform/AWS/VPC/infrastructure.tf
- Resource: **aws_security_group**
- Rule:

  ```
  ingress {
    from_port = 563
    to_port = 563
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ```

E

Deploying on a Preexisting Cluster

ICM provides you with the option of allocating your own cloud or virtual compute nodes or physical servers to deploy containers on. The provisioning phase usually includes allocation and configuration subphases, but when the Provider field is set to PreExisting, ICM bypasses the allocation phase and moves directly to configuration. There is no unprovisioning phase for a preexisting cluster.

- Compute Node Requirements
- Definitions File

E.1 Compute Node Requirements

The preexisting compute nodes must satisfy the criteria listed in the following sections.

- Operating System
- SSH
- Ports
- Storage Volumes

E.1.1 Operating System

Supported operating systems include:

- Red Hat Enterprise Linux 7.2 or later
- Ubuntu 18.04 or later.

The following operating systems are not supported but have been tested:

- CentOS
- SUSE Enterprise Linux 12.3 or later

E.1.2 SSH

ICM requires that SSH be installed and the SSH daemon running.

Additionally, a nonroot account must be specified in the SSHUser field in the defaults file. This account should have the following properties:

- It must provide **sudo** access without requiring a password. You can enable this by creating or modifying a file in /etc/sudoers.d/ to contain the following line:

  ```
  <accountname> ALL=(ALL) NOPASSWD:ALL
  ```

- If the home directory is located anywhere other than /home, it should be specified in the Home field in the defaults file, for example:

  ```
  "Home": "/users/"
  ```

  Note that the home directory must not be a network directory shared among nodes (for example /nethome), because this would cause configuration files to overwrite one another.

ICM can log in as SSHUser using SSH keys or password login. Even if password logins are enabled, ICM will always try to log in using SSH first.

If you've configured your machines with SSH keys, you must specify the SSH public/private key pair your configuration file using the SSHPublicKey and SSHPrivateKey fields.

During the configuration phase, ICM configures SSH login and disables password login by default. If you don't wish password login to be disabled, you can **touch** the following sentinel file in the home directory of the SSHUser account:

```
mkdir -p ICM
touch ICM/disable_password_login.done
```

If you've configured your machines with a password, specify it using the SSHPassword field in your configuration file. ICM assumes these credentials are insecure.

Enabling password login and specifying the SSHPassword field does not remove the requirement that ICM be able to carry out all postconfiguration operations via SSH.

E.1.3 Ports

To avoid conflicting with local security policies and because of variations among operating systems, ICM does not attempt to open any ports. The following table contains the default ports that must be opened to make use of various ICM features. As described in General Parameters, the ports are configurable, for example:

```
"JDBCGatewayPort": "62975"
```

If you change one or more from the defaults in this way, you must ensure that the ports you specify are open.

Port	Protocol	Service	Notes
22	tcp	SSH	Required.
2376	tcp	Docker (TLS mode)	Required.
80	tcp	Web	Required to access the public Apache web server on nodes of role WS (web server).
53	tcp udp	DNS	Required for Weave DNS.
6783	tcp udp	Weave Net	Required for Overlay=Weave (default for all providers except PreExisting).
4040	tcp	Weave Scope	Required for Weave monitoring.

Port	Protocol	Service	Notes
7077 7000 8080 8081 6066 7001 7005	tcp	Spark	Required for web access to InterSystems IRIS+Spark containers. Different ports may be specified using the fields SparkMasterPort, SparkWorkerPort, SparkMasterWebUIPort, SparkWorkerWebUIPort, SparkRESTPort, SparkDriverPort, and SparkBlocKManagerPort, respectively.
7077 8080 8081	tcp	Spark	Required for web access to Spark containers. Different ports may be specified using the Spark*Port fields.
51773	tcp	InterSystems IRIS™ Superserver	Required. A different port may be specified using the SuperServerPort field.
52773	tcp	InterSystems IRIS Webserver	Required. A different port may be specified using the WebServerPort field.
2188	tcp	InterSystems IRIS ISCAgent	Required for mirroring.A different port may be specified using the ISCAgentPort field.
4002	tcp	InterSystems IRIS License Server	Required Note: A different port may be specified using the LicenseServerPort field.
53773	tcp	InterSystems IRIS JDBC Gateway Port	Required to use the JDBC Gateway. A different port may be specified using the JDBCGatewayPort field.

E.1.4 Storage Volumes

As described in Storage Volumes Mounted by ICM, ICM mounts storage volumes used by InterSystems IRIS and Docker under /dev, using names specified by the fields listed in Device Name Parameters. These fields have defaults for other providers, but not for PreExisting, so they must be included in your defaults file for PreExisting deployments.

For provider PreExisting, the device name **null** (literal string) changes the behavior as follows:

- DockerDeviceName:

  Docker is configured to use the loopback logical volume manager (see Configure loop-lvm mode for testing); this mode is not suitable for production use.

- DataDeviceName, WIJDeviceName, Journal1DeviceName, Journal2DeviceName:

  ICM simply creates the mount point as a local directory on the host volume; this mode is not suitable for production use.

E.2 Definitions File

The main difference between PreExisting and the other providers is the contents of the definitions file, which contains exactly one entry per node, rather than one entry per role with a Count field to specify the number of nodes of that role.

Each node is identified by its IP address or fully-qualified domain name. The fields shown in the following table are required for each node definition in a preexisting cluster deployment (along with other required fields described in other sections of this document):

Parameter	Description	Example
IPAddress	IP address of the preexisting node.	172.16.110.9
DNSName	Fully-qualified DNS name of the preexisting node; can be used in place of IPAddress. Must be resolvable by ICM and within the cluster. (For all other providers this is an output field, rather than an input field.)	subnet2node21.mycointernal.com
SSHUser	Nonroot user with passwordless **sudo** access.	icmuser