InterSystems Cloud Manager Guide

Version 2020.3
2021-02-04

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

InterSystems Cloud Manager Guide
InterSystems IRIS Data Platform Version 2020.3 2021-02-04

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Document **1**

1 ICM Overview **3**
- 1.1 Benefits of InterSystems Cloud Manager 3
- 1.2 The InterSystems Cloud Manager Application Lifecycle 5
  - 1.2.1 Define Goals 6
  - 1.2.2 Provision 7
  - 1.2.3 Deploy 7
  - 1.2.4 Manage 8

2 Essential InterSystems Cloud Manager Elements **9**
- 2.1 InterSystems Cloud Manager Image 9
- 2.2 Provisioning Platforms 9
- 2.3 Deployment Platforms 10
- 2.4 Defining Nodes in the Deployment 10
- 2.5 Field Values 11
- 2.6 Command Line 12
- 2.7 Configuration, State, and Log Files 13
  - 2.7.1 The Definitions File 14
  - 2.7.2 The Defaults File 14
  - 2.7.3 The Instances File 15
  - 2.7.4 The State Directory and State Files 15
  - 2.7.5 Log Files and Other InterSystems Cloud Manager Files 16
- 2.8 Docker Repositories 16
  - 2.8.1 Logging Into a Docker Repository 16
  - 2.8.2 Setting Up a Docker Repository 17

3 Using InterSystems Cloud Manager **19**
- 3.1 ICM Use Cases 19
- 3.2 Launch ICM 20
  - 3.2.1 Downloading the ICM Image 20
  - 3.2.2 Running the ICM Container 21
  - 3.2.3 Upgrading an ICM Container 22
- 3.3 Obtain Security-Related Files 22
  - 3.3.1 Cloud Provider Credentials 22
  - 3.3.2 SSH and TLS Keys 22
- 3.4 Define the Deployment 23
  - 3.4.1 Shared Defaults File 24
  - 3.4.2 Distributed Cache Cluster Definitions File 28
  - 3.4.3 Sharded Cluster Definitions File 30
  - 3.4.4 Customizing InterSystems IRIS Configurations 32
- 3.5 Provision the Infrastructure 32
  - 3.5.1 The icm provision Command 33
  - 3.5.2 Reprovisioning the Infrastructure 34
  - 3.5.3 Infrastructure Management Commands 36
- 3.6 Deploy and Manage Services 38
  - 3.6.1 The icm run Command 38
  - 3.6.2 Redeploying Services 42
  - 3.6.3 Container Management Commands 43

3.6.4 Service Management Commands 45
3.7 Unprovision the Infrastructure 49

4 ICM Reference 51
4.1 ICM Commands and Options 51
4.2 ICM Configuration Parameters 54
4.2.1 General Parameters 55
4.2.2 Security-related Parameters 60
4.2.3 Port and Protocol Parameters 64
4.2.4 CPF Parameters 66
4.2.5 Provider-Specific Parameters 66
4.2.6 Device Name Parameters 77
4.2.7 Alphabetical List of User Parameters 77
4.3 ICM Node Types 82
4.3.1 Role DATA: Sharded Cluster Data Node 83
4.3.2 Role COMPUTE: Sharded Cluster Compute Node 84
4.3.3 Role DM: Distributed Cache Cluster Data Server, Standalone Instance, Shard Master Data Server 84
4.3.4 Role DS: Shard Data Server 84
4.3.5 Role QS: Shard Query Server 84
4.3.6 Role AM: Distributed Cache Cluster Application Server, Shard Master Application Server 85
4.3.7 Role AR: Mirror Arbiter 85
4.3.8 Role WS: Web Server 85
4.3.9 Role SAM: System Alerting and Monitoring Node 86
4.3.10 Role LB: Load Balancer 86
4.3.11 Role VM: Virtual Machine Node 88
4.3.12 Role CN: Container Node 88
4.3.13 Role BH: Bastion Host 88
4.4 ICM Cluster Topology and Mirroring 88
4.4.1 Rules for Mirroring 89
4.4.2 Nonmirrored Configuration Requirements 90
4.4.3 Mirrored Configuration Requirements 92
4.5 Storage Volumes Mounted by ICM 95
4.6 InterSystems IRIS Licensing for ICM 96
4.7 ICM Security 97
4.7.1 Host Node Communication 97
4.7.2 Docker 97
4.7.3 Weave Net 97
4.7.4 InterSystems IRIS 98
4.7.5 Private Networks 98
4.8 Deploying with Customized InterSystems IRIS Configurations 99
4.9 Deploying Across Multiple Zones 100
4.10 Deploying Across Multiple Regions or Providers 101
4.10.1 Deploying Across Multiple Regions on GCP 102
4.10.2 Deploying Across Multiple Regions on Azure 103
4.10.3 Deploying Across Multiple Regions on AWS and Tencent 105
4.11 Deploying on a Private Network 109
4.11.1 Deploy Within an Existing Private Network 109
4.11.2 Deploy on a Private Network Through a Bastion Host 111
4.12 Deploying InterSystems API Manager 113

4.13 Monitoring in ICM 114
4.13.1 System Alerting and Monitoring 114
4.13.2 Deploying Third-party Monitoring with ICM 115
4.14 ICM Troubleshooting 115
4.14.1 Host Node Restart and Recovery 115
4.14.2 Correcting Time Skew 117
4.14.3 Timeouts Under ICM 117
4.14.4 Docker Bridge Network IP Address Range Conflict 118
4.14.5 Weave Network IP Address Range Conflict 118
4.14.6 Huge Pages 118

Appendix A: Containerless Deployment 121
A.1 Containerless Deployment Platforms 121
A.2 Enabling Containerless Mode 121
A.3 Installing InterSystems IRIS 122
A.4 Reinstalling InterSystems IRIS 123
A.5 Uninstalling InterSystems IRIS 124
A.6 Additional Containerless Mode Commands 124

Appendix B: Sharing ICM Deployments 127
B.1 Sharing Deployments in Distributed Management Mode 127
B.1.1 Distributed Management Mode Overview 127
B.1.2 Configuring Distributed Management Mode 128
B.1.3 Upgrading ICM Using Distributed Management Mode 129
B.2 Sharing Deployments Manually 130
B.2.1 State Files 130
B.2.2 Maintaining Immutability 130
B.2.3 Persisting State Files 131

Appendix C: Scripting with ICM 133
C.1 ICM Exit Status 133
C.2 ICM Logging 133
C.3 Remote Script Invocation 134
C.4 Using JSON Mode 134
C.4.1 Normal Output 134
C.4.2 Abnormal Output 137

Appendix D: Using ICM with Custom and Third-Party Containers 139
D.1 Container Naming 139
D.2 Overriding Default Commands 139
D.3 Using Docker Options 140
D.3.1 Restarting 140
D.3.2 Privileges 140
D.3.3 Environment Variables 140
D.3.4 Mount Volumes 140
D.3.5 Ports 141

Appendix E: Deploying on a Preexisting Cluster 143
E.1 Host Node Requirements for PreExisting 143
E.1.1 SSH 143
E.1.2 Ports 144
E.1.3 Storage Volumes 145
E.2 Definitions File for PreExisting 145

List of Figures

Figure 1–1: ICM Makes It Easy 4
Figure 1–2: Containers Support the DevOps Approach 5
Figure 1–3: Role of ICM in the Application Lifecycle 6
Figure 2–1: ICM Configuration Files Define Deployments 12
Figure 3–1: Distributed Cache Cluster to be Deployed by ICM 29
Figure 3–2: Sharded Cluster to be Deployed by ICM 31
Figure 3–3: Interactive ICM Commands 46
Figure 4–1: ICM Nonmirrored Topologies 91
Figure 4–2: ICM Mirrored Topologies 94
Figure 4–3: ICM Deployed within Private Subnet 109
Figure 4–4: ICM Deployed Outside a Private Network with a Bastion Host 112

List of Tables

Table 4–1: ICM Commands 51
Table 4–2: ICM Command-Line Options 52
Table 4–3: ICM Node Types 82

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:

- InterSystems Cloud Manager Overview

  Provides an overview of ICM's purpose, design, uses, and benefits.

- Essential InterSystems Cloud Manager Elements

  Describes the elements involved in using ICM, including Docker images and repositories, platforms, and configuration files.

- Using InterSystems Cloud Manager

  Provides detailed step-by-step procedures for using ICM to provision infrastructure and deploy services.

- InterSystems Cloud Manager Reference

  Provides reference information for ICM including commands, options and configuration parameters; node types and topology; licensing and security; multizone, multiregion, and private network deployment; monitoring; and troubleshooting.

This book also contains the following appendixes:

- Containerless Deployment

  Explains how to use ICM to deploy noncontainerized services on provisioned infrastructure.

- Sharing InterSystems Cloud Manager Deployments

  Describes ICM's distributed management mode, which lets you manage the infrastructure and services in a single deployment from multiple ICM containers.

- Scripting with InterSystems Cloud Manager

  Explains how to use scripts to provision infrastructure and deploy services with ICM.

- Using InterSystems Cloud Manager with Custom and Third-Party Containers

  Provides instructions for using ICM to deploy services in custom and third-party containers.

- Deploying on a Preexisting Cluster

  Provides instructions for using ICM to deploy services on preexisting physical or virtual infrastructure.

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.

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

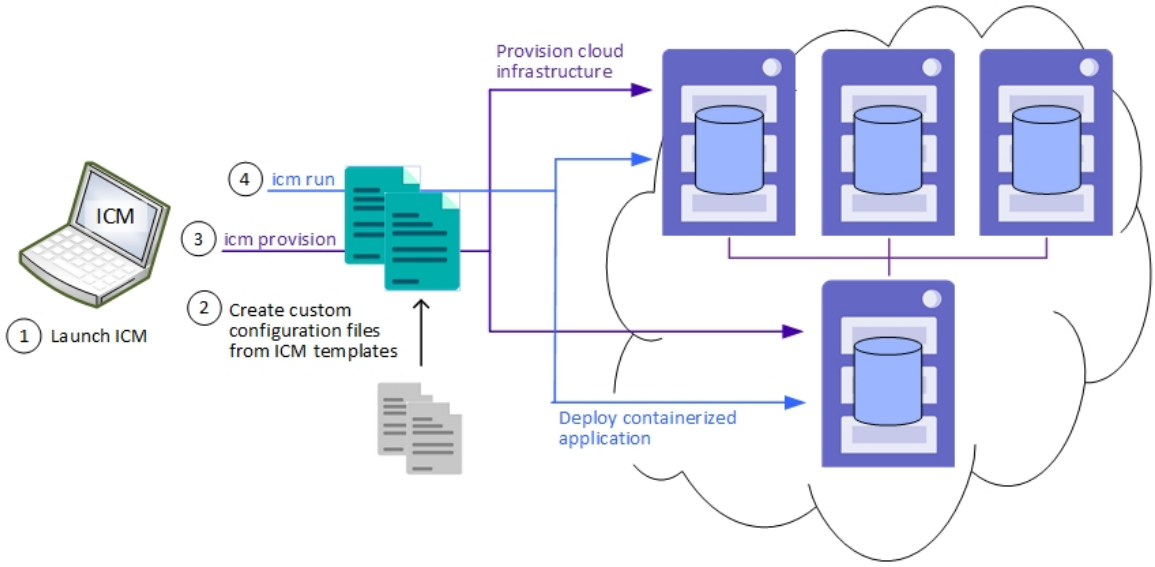
1.1 Benefits of InterSystems Cloud Manager

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 Cloud Platform, Amazon Web Services, Microsoft Azure, and Tencent Cloud. 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



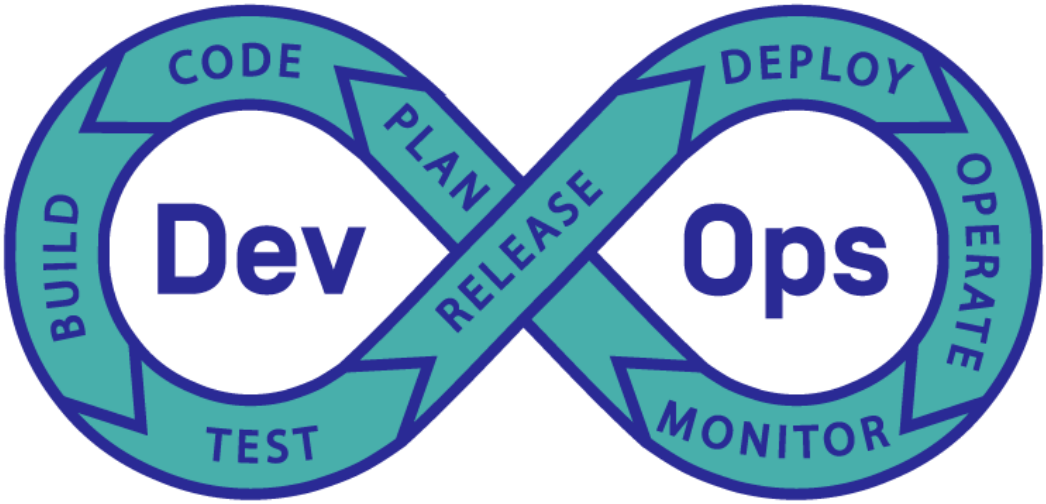
Using ICM lets you take advantage of the efficiency, agility, and repeatability provided by virtual and cloud computing and containerized software without major development or retooling. The InterSystems IRIS configurations ICM can provision and deploy range from a stand-alone instance, through load-balanced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

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 host 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



ICM exploits these advantages to bring you the following benefits:

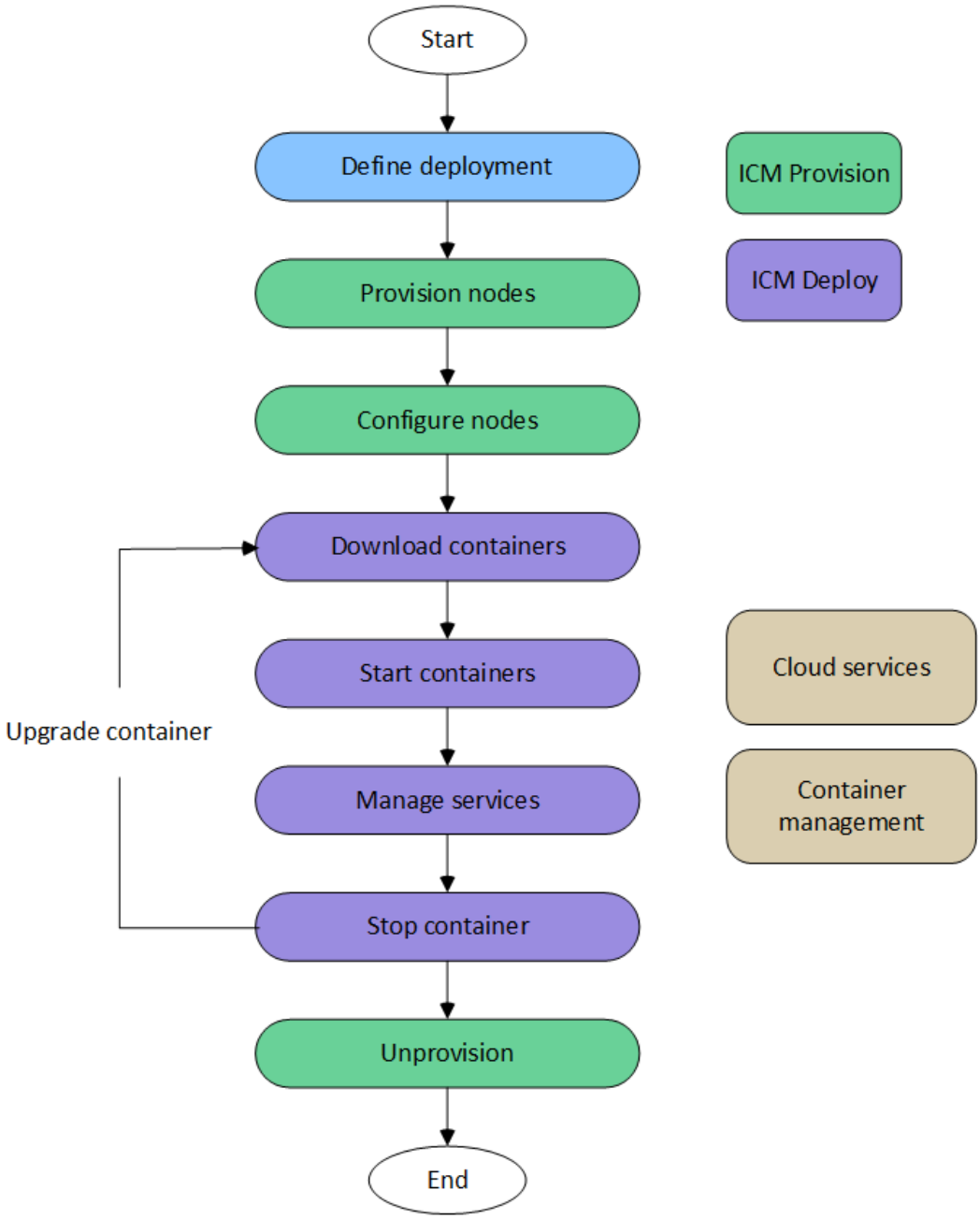
- Automated provisioning and deployment, and command-line management, of large-scale, cloud-based InterSystems IRIS configurations.
- Integration of existing InterSystems IRIS and InterSystems IRIS-based applications into your enterprise's DevOps toolchain.
- Stability, robustness, and minimization of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

If you prefer not to work with Docker containers, you can use ICM to provision cloud infrastructure and install noncontainerized InterSystems IRIS instances on that infrastructure, or to install InterSystems IRIS on existing infrastructure. For more information about using ICM's containerless mode, see the appendix Containerless Deployment.

1.2 The InterSystems Cloud Manager 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, TLS) , InterSystems IRIS licenses, types and sizes of the host nodes you want, and so on. (See Define the Deployment for details.)

Note: In this document, the term *host 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 four main provisioning activities: creating (provisioning), configuring, modifying, and detroying (unprovisioning) host 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
Executes provisioning tasks only, cannot integrate provisioning with deployment and configuration	Coordinates all phases, including in elastic reprovisioning and redeployment (for example adding nodes to the cluster infrastructure, then deploying and configuring InterSystems IRIS on the nodes to incorporate them into the cluster)
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 InterSystems IRIS images in Docker containers on the host 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.
- ICM can redeploy services on infrastructure that has been modified since the initial deployment, including upgrading or adding new containers.

Note: For a brief introduction to the use of InterSystems IRIS in containers, including a hands-on experience, see *First Look: InterSystems Products in Containers*; for detailed information about deploying InterSystems IRIS and InterSystems IRIS-based applications in Docker containers using methods other than ICM, see *Running InterSystems Products 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 InterSystems Cloud Manager Elements

The chapter describes the essential elements involved in using ICM.

2.1 InterSystems Cloud Manager 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 provisioning platform on which you intend to provision infrastructure and deploy containers, or the existing infrastructure on which you intend to deploy.

2.2 Provisioning Platforms

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

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure)
- Tencent Cloud (Tencent)
- 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.

On AWS, GCP, Azure, and Tencent, ICM can provision and deploy a single configuration across multiple zones within a region, across multiple regions, or even across cloud provider platforms.

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

Container images from InterSystems comply with the Open Container Initiative (OCI) specification, and are built using the Docker Enterprise Edition engine, which fully supports the OCI standard and allows for the images to be certified and featured in the Docker Hub registry.

InterSystems images are built and tested using the widely popular container Ubuntu operating system, and ICM therefore supports their deployment on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.

2.4 Defining Nodes in the Deployment

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.

In preparing to use ICM, you must define your target configuration (see Define the Deployment in the "Using ICM" chapter) 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 data nodes and (optionally) compute nodes will be included in the cluster, and whether the data nodes 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 four data nodes and eight compute nodes:

```
[
    {
        "Role": "DATA",
        "Count": "4",
        "LicenseKey": "ubuntu-sharding-iris.key"
    },
    {
        "Role": "COMPUTE",
        "Count": "8",
        "StartCount": "5",
        "LicenseKey": "ubuntu-sharding-iris.key"
    }
]
```

The Mirror field, which determines whether the data nodes are mirrored, appears in the defaults file. Mirror deployment is covered at length in ICM Cluster Topology and Mirroring.

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.

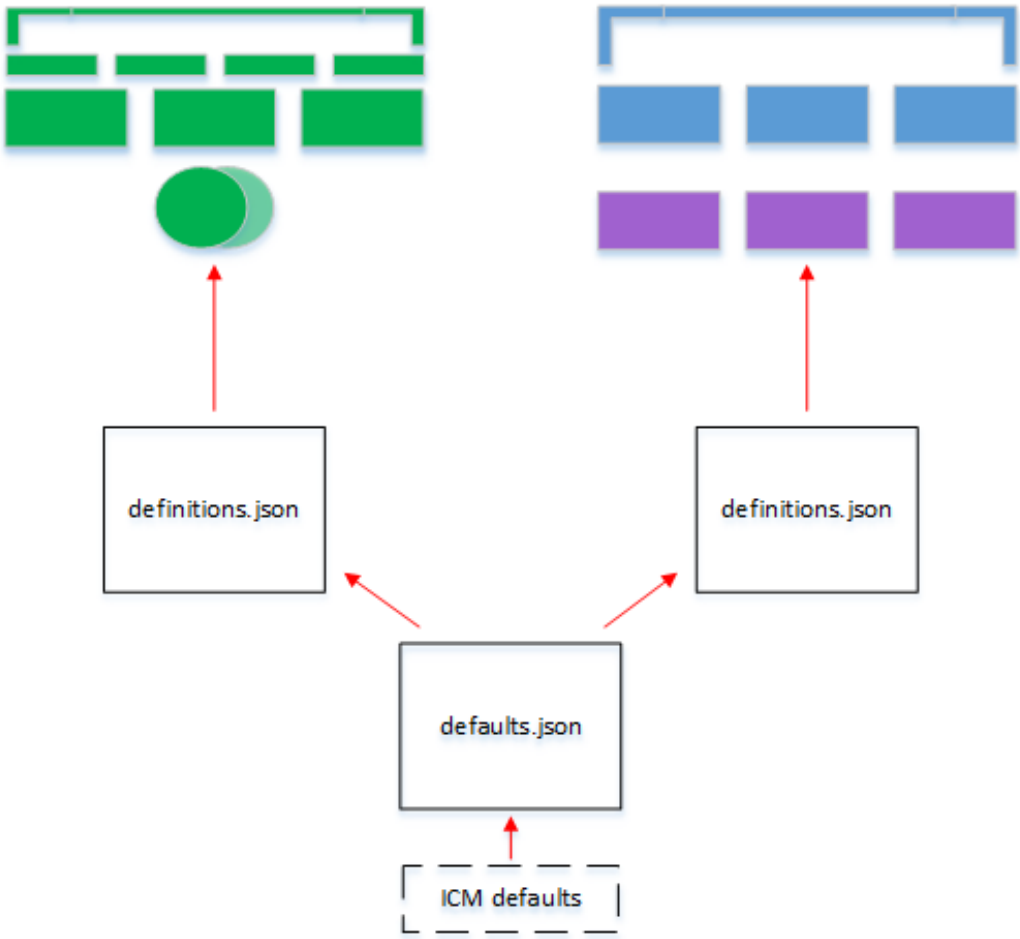
There are also defaults for most ICM fields. In descending order of precedence, these specifications are ranked as follows:

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 two deployments sharing a single defaults.json file, with different definitions.json files — the first for a distributed cache cluster, the second for a sharded cluster.

Figure 2–1: ICM Configuration Files Define Deployments



For comprehensive lists of the required and optional fields that can be specified in these files, see ICM Configuration Parameters in the "ICM Reference" chapter.

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 DATA -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 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 provisioning platform to use and 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. (For comprehensive lists of the required and optional fields that can be specified in these files, see ICM Configuration Parameters in the "ICM Reference" chapter.)
- 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 host nodes to be provisioned for a particular deployment. ICM uses the definitions file found in the current working directory.

The definitions 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, Tencent, VMware vSphere, or PreExisting).

Most fields can appear in this file, repeated as needed 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. Other fields that cannot be included in the definitions.json file and will cause an error if they are included are Label and Tag.

Fields that vary between node types (for example, Role) must be included in the node definitions in definitions.json. The definitions.json is also used to override either ICM defaults or settings in the defaults.json file for specific node types. For instance, in the following example, which shows the contents of a sample definitions.json file for provisioning a distributed cache cluster consisting of a mirrored data server (**"Role": "DM"**"), three application servers (**"Role": "AM"**), and a mirror arbiter node (**"Role": "AR"**) on AWS:

- The DataVolumeSize field appears only for the DM nodes 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 an arbiter container is to 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, the AM node definition here includes "LoadBalancer": "true" to automatically provision a load balancer for the AM nodes. This setting can also be used with WS nodes, but applying it 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": "6",
        "InstanceType": "t2.small",
        "DockerImage": "intersystems/arbiter:2020.3.0.221.0"
    }
]
```

By modifying the definitions.json file, then reprovisioning and/or redeploying, you can elastically scale and alter an existing configuration by adding or removing nodes or modifying existing nodes.

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. ICM uses the defaults file found in the current working directory.

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 to be used with the definitions.json file shown 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": "ubuntu",
    "SSHPublicKey": "/Samples/ssh/insecure-ssh2.pub",
    "SSHPrivateKey": "/Samples/ssh/insecure",
    "DockerImage": "intersystems/iris:2020.3.0.221.0",
    "DockerUsername": "xxxxxxxxxxxx",
    "DockerPassword": "xxxxxxxxxxxx",
    "DockerVersion": "18.06.1~ce~3-0~ubuntu",
    "TLSKeyDir": "/Samples/tls/",
    "LicenseDir": "/Samples/license/",
    "Region": "us-west-1",
    "Zone": "us-west-1c",
    "AMI": "ami-c509eda6",
    "InstanceType": "m4.large",
    "Credentials": "/Samples/AWS/sample.credentials",
    "ISCPassword": "",
    "Mirror": "false",
    "UserCPF": "/Samples/cpf/iris.cpf"
}
```

2.7.3 The Instances File

The instances file (instances.json), generated by ICM during the provisioning phase, describes the set of host 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. The instances file is created in the current working directory.

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 the state subdirectory, for use during the lifetime of the provisioned infrastructure. The state subdirectory is created in the current working directory. by default.

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:

```
./state/Acme-DM-TEST
./state/Acme-AM-TEST
./state/Acme-AR-TEST
```

If you prefer to create your own state directory with a different name or in a different location, you can use the **-stateDir** command line option with the **icm provision** command to override the default location, but you must then continue using the **-stateDir** option to specify that location in all subsequent provisioning commands, for example when you unprovision the infrastructure.

Important: ICM relies on the state files it creates for accurate, up to date information about the infrastructure it has provisioned; 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.

2.7.5 Log Files and Other InterSystems Cloud Manager 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 at docker.com.

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

  For information about using the InterSystems Container Registry (ICR), see Downloading the ICM Image in the "Using ICM" chapter.

- 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 InterSystems Cloud Manager

BOBA

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

- Review deployments based on two *sample use cases* that are used as examples throughout the chapter.
- Use the **docker run** command with the ICM image provided by InterSystems to start the ICM container and open a command line.
- Obtain the cloud provider and TLS credentials needed for secure communications by ICM.
- Decide how many of each nodes type you want to provision and make other needed choices, then create the defaults.json and definitions.json configuration files needed for the deployment.
- Use the **icm provision** command to provision the infrastructure and explore ICM's infrastructure management commands. You can reprovision at any time to modify existing infrastructure, including scaling out or in.
- Run the **icm run** command to deploy containers, and explore ICM's container and service management commands. You can redeploy when the infrastructure has been reprovisioned or when you want to add, remove, or upgrade containers.
- 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 — Three mirrored data nodes, arbiter node, and load balancer. 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 (The icm provision Com-

mand) is for the distributed cache cluster; output shown for the deployment phase (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.

- Downloading the ICM Image
- Running the ICM Container
- Upgrading an 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.

Note: Multiple ICM containers can be used to manage a single deployment, for example to make it possible for different people to execute different phases of the deployment process; for detailed information, see the appendix "Sharing ICM Deployments."

3.2.1 Downloading the ICM Image

To use ICM, you need to download the ICM image to the system you are working on; this requires you to identify the registry from which you will download it and the credentials you need for access. Similarly, for ICM to deploy InterSystems IRIS and other InterSystems components, it requires this information for the images involved. The registry from which ICM downloads images must be accessible to the cloud provider you use (that is, not behind a firewall), and for security must require ICM to authenticate using the credentials you provide to it.

The InterSystems Container Registry (ICR) at containers.intersystems.com includes repositories for all images available from InterSystems, including ICM and InterSystems IRIS images; Using the InterSystems Container Registry provides detailed information about the available images and how to use the ICR. In addition, your organization may already store InterSystems images in its own private image registry; if so, obtain the location and the credentials needed to authenticate from the responsible administrator.

Once you are logged into the ICR or your organization's registry, you can use the **docker pull** command to download the image; the following example shows a pull from the ICR.

```
$ docker login containers.intersystems.com
Username: pmartinez
Password: **********
$ docker pull containers.intersystems.com/intersystems/icm:2020.3.0.221.0
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
$ docker images
REPOSITORY                                      TAG            IMAGE ID      CREATED      SIZE
containers.intersystems.com/intersystems/iris 2020.3.0.221.0 15627fb5cb76  1 minute ago 1.39GB
containers.intersystems.com/intersystems/sam  1.0.0.115      15627fb5cb76  3 days ago   1.33GB
acme/centos                                     7.3.1611       262f7381844c  2 weeks ago  192MB
acme/hello-world                                latest         05a3bd381fc2  2 months ag  1.84kB
```

For simplicity, the instructions in this document assume you are working with the InterSystems images from an **`InterSystems`** repository a **`2020.3.0.221.0`** tag, for example **`intersystems/icm:2020.3.0.221.0`**.

Whether the images used by ICM are in the same registry or a different one, you will need to provide the image identifier, including registry, in the DockerImage field and the credentials needed to authenticate in the DockerUsername and DockerPassword fields for each image, as described in Docker Repositories in the "Elements of ICM" chapter.

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; if it is present, it is updated if necessary (this step 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:2020.3.0.221.0
```

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 (see Manage data in Docker in the Docker documentation) or copy files into the ICM 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:2020.3.0.221.0
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). When deploying InterSystems IRIS containers, ICM automatically formats, partitions, and mounts several storage volumes; for more information, see Storage Volumes Mounted by ICM in the "ICM Reference" chapter.

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 additional requirements and general information about Docker for Windows.

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.

3.2.3 Upgrading an ICM Container

Distributed management mode, which allows different users on different systems to use ICM to manage with the same deployment, provides a means of upgrading an ICM container while preserving the needed state files of the deployment it is managing (see The State Directory and State Files). Because this is the recommended way to upgrade an ICM container that is managing a deployment, you may want to configure distributed management mode each time you use ICM, whether you intend to use distributed management or not, so that this option is available. For information about upgrading ICM in service discovery mode, see Upgrading ICM Using Distributed Management Mode.

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 of the "ICM Reference" chapter. In the ICM configuration files, you identify the location of these credentials using the parameter(s) specific to the provider; for AWS, this is 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 TLS Keys

ICM uses SSH to provide secure access to the operating system of provisioned nodes, and TLS to establish secure connections to Docker, InterSystems Web Gateway, and JDBC, and between nodes in InterSystems IRIS mirrors, distributed cache clusters, and sharded clusters. 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 host 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 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. You can begin with the template defaults.json and definitions.json files provided within in the ICM container in the /Samples directory tree, for example /Samples/AWS.

As noted in Configuration, State and Log Files, defaults.json is often used to provide shared settings for multiple deployments in a particular category, for example those that are provisioned on the same platform, while separate definitions.json files define the node types that must be provisioned and configured for each deployment. For example, the separate definitions files illustrated here define the two target deployments described at the start of this chapter: the distributed cache cluster includes two DM nodes as a mirrored data server, three load-balanced AM nodes as application servers, and an arbiter (AR) node, while the sharded cluster includes six DATA nodes configured as three load-balanced mirrored data nodes, plus an arbiter node. At the same time, the deployments can share a defaults.json file because they have a number of characteristics in common; for example, they are both on AWS. use the same credentials, provision in the same region and availability zone, and deploy the same InterSystems IRIS image.

While some fields (such as Provider) must appear in defaults.json and some (such as Role) in definitions.json, others can be used in either depending on your needs. In this case, for example, the InstanceType field appears in the shared defaults file and both definitions files, because the DM, AM, DATA, and AR nodes all require different compute resources; for this reason a single defaults.json setting, while establishing a default instance type, is not sufficient.

The following sections explain how you can customize the configuration of the InterSystems IRIS instances you deploy and review the contents 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.

- Shared Defaults File
- Distributed Cache Cluster Definitions File
- Sharded Cluster Definitions File
- Customizing InterSystems IRIS Configurations

Bear in mind that ICM allows you to modify the definitions file of an existing deployment and then reprovision and/or redeploy to add or remove nodes or to alter existing nodes. For more information, see Reprovisioning the Infrastructure and Redeploying Services.

Important: Both field names and values are case-sensitive; for example, to select AWS as the cloud provider you must include "Provider":"AWS" in the defaults file, not "provider":"AWS", "Provider":"aws", and so on.

Note: The fields included here represent a subset of the potentially applicable fields; see ICM Configuration Parameters for comprehensive lists of all required and optional fields, both general and provider-specific.

3.4.1 Shared Defaults File

The field/value pairs shown in the table in this section represent the contents of a defaults.json file that can be used for both the distributed cache cluster deployment and the sharded cluster deployment. As described at the start of this section, this file can be created by making a few modifications to the /Samples/AWS/default.json file, which is illustrated in the following:

```
{
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "TEST",
    "DataVolumeSize": "10",
    "SSHUser": "ubuntu",
    "SSHPublicKey": "/Samples/ssh/insecure-ssh2.pub",
    "SSHPrivateKey": "/Samples/ssh/insecure",
    "DockerImage": "intersystems/iris:2020.3.0.221.0",
    "DockerUsername": "xxxxxxxxxxxx",
    "DockerPassword": "xxxxxxxxxxxx",
    "TLSKeyDir": "/Samples/tls/",
    "LicenseDir": "/Samples/license/",
    "Region": "us-west-1",
    "Zone": "us-west-1c",
    "DockerVersion": "ce-18.09.8.ce",
    "AMI": "ami-c509eda6",
    "InstanceType": "m4.large",
    "Credentials": "/Samples/AWS/sample.credentials",
    "ISCPassword": "",
    "Mirror": "false",
    "UserCPF": "/Samples/cpf/iris.cpf"
}
```

The order of the fields in the table matches the order of this sample defaults file.

In the defaults file for a different provider, some of the fields have different provider-specific values, while others are replaced by different provider-specific fields. For example, in the Tencent defaults file, the value for InstanceType is S2.MEDIUM4, a Tencent-specific instance type that would be invalid on AWS, while the AWS AMI field is replaced by the equivalent Tencent field, ImageID. You can review these differences by examining the varying defaults.json files in the /Samples directory tree and referring to the General Parameters and Provider-Specific Parameters tables in the "ICM Reference" chapter.

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 replaced by internal paths to external storage volumes mounted when the ICM container is run, as described in the Launch ICM. For additional information about these files, see ICM Security and Security-Related Parameters in the "ICM Reference" chapter.

Shared characteristic	/Samples/AWS/defaults.json	Customization example	Customization explanation
Platform to provision infrastructure on, in this case Amazon Web Services; see Provisioning Platforms in the "Essential ICM Elements" chapter.	"Provider": "AWS",	n/a	If value is changed to GCP, Azure, Tencent, vSphere, or PreExisting, different fields and values from those shown here are required.

Shared characteristic	/Samples/AWS/defaults.json	Customization example	Customization explanation
Naming pattern for provisioned nodes is *Label-Role-Tag-NNNN*, where *Role* is the value of the Role field in the definitions file, for example ANDY-DATA-TEST-0001. Modify these so that node names indicate ownership and purpose.	"Label": "Sample", "Tag": "TEST",	"Label": "ANDY", "Tag": "TEST",	Update to identify owner.
Size (in GB) of the persistent data volume to create for InterSystems IRIS containers; see Storage Volumes Mounted by ICM in the "ICM Reference" chapter. Can be overriden for specific node types in the definitions file.	"DataVolumeSize": "10",	"DataVolumeSize":"250",	If all deployments using the defaults file consist of sharded cluster (DATA) nodes only, enlarging the default size of the data volume is recommended.
Nonroot account with **sudo** access, used by ICM for SSH access to provisioned nodes. On AWS, the required value depends on the AMI; see Security-Related Parameters in the "ICM Reference" chapter.	"SSHUser": "ubuntu",	"SSHUser": "ec2–user",	Change to this account if specifying a Red Hat Enterprise Linux AMI, rather than Ubuntu.
Locations of needed security key files; see Obtain Security-Related Files and Security-Related Parameters. Because provider is AWS, the SSH2–format public key in /Samples/ssh/ is specified.	"SSHPublicKey": "/Samples/ssh/secure-ssh2.pub", "SSHPrivateKey": "/Samples/ssh/secure-ssh2", "TLSKeyDir": "/Samples/tls/",	"SSHPublicKey": "/mydir/keys/mykey.pub", "SSHPrivateKey": "/mydir/keys/mykey.ppk", "TLSKeyDir": "/mydir/keys/tls/",	If you stage your keys on a mounted external volume, update the paths to reflect this.

Shared characteristic	/Samples/AWS/defaults.json	Customization example	Customization explanation
The Docker version to be installed on provisioned nodes; typically you can keep the default value.	"DockerVersion": "18.06.1~ce~3-0~ubuntu",	"DockerVersion": "18.06.1~ce~3-0~ubuntu",	The version in each /Samples/.../defaults.json is generally correct for the platform. However. if your organization uses a different version of Docker, you may want that version installed on the cloud nodes instead.
The Docker image to deploy on provisioned nodes; see Docker Repositories in the "Elements of ICM" chapter, The icm run Command, and General Parameters in the "ICM Reference" chapter. This field can also be included in a node definition in definitions.json, overriding the defaults file value, as illustrated in Distributed Cache Cluster Definitions File.	"DockerImage": "intersystems/iris:2020.3.0.221.0",	"DockerImage": "acme/iris:2020.3.0.221.0"	If you pushed the InterSystems IRIS image to your organization's registry, update the image spec. *Note*: InterSystems IRIS images for standard platforms are named **`iris`**; those for ARM platforms are named **`iris-arm64`**.
Credentials to log into the Docker registry in which the image specified by the previous field is stored; see Downloading the ICM Image.	"DockerUsername": "...", "DockerPassword": "...",	"DockerUsername": "AndyB", "DockerPassword": "*password*",	Update to use your Docker credentials for the specified registry.

Shared characteristic	/Samples/AWS/defaults.json	Customization example	Customization explanation
Location of InterSystems IRIS license keys staged in the ICM container and individually specified by the LicenseKey fields in the definitions file; see InterSystems IRIS Licensing for ICM in the “ICM Rference” chapter.	“LicenseDir”: “/Samples/Licenses”,	“LicenseDir”: “/mydir/licenses”,	If you stage your licenses on a mounted external volume, update the paths to reflect this.
Geographical region of provider’s compute resources in which infrastructure is to be provisioned; see General Parameters.	"Region": "us-west-1",	"Region": "us-east-2",	If you want to provision in another valid combination of region and availability zone, update the values to reflect this.
Availability zone within specified region in which to locate provisioned nodes; see General Parameters.	"Zone": "us-west-1c",	"Zone": "us-east-2a",	
AMI to use as platform and OS template for nodes to be provisioned; see Amazon Web Services (AWS) Parameters in the “ICM Reference” chapter.	"AMI": "ami-c509eda6",	“AMI”: “ami-e24b7d9d”,	If you want to provision from another valid combination of AMI and instance type, update the values to reflect this.
Instance type to use as compute resources template for nodes to be provisioned; see Amazon Web Services (AWS) Parameters.	"InstanceType": "m4.large",	"InstanceType": "m5ad.large",	
Credentials for AWS account; see Amazon Web Services (AWS) Parameters.	"Credentials": "/Samples/AWS/credentials",	“Credentials”: “/mydir/aws-credentials”,	If you stage your credentials on a mounted external volume, update the path to reflect this.

Shared characteristic	/Samples/AWS/defaults.json	Customization example	Customization explanation
Password for deployed InterSystems IRIS instances. Recommended approach is to specify on the deployment command line (see Deploy and Manage Services) to avoid displaying password in a configuration file	“ISCPassword”: "",	(delete)	Remove in favor of specifying password by using **-password** option of **icm run** command.
Whether specific node types (including DM and DATA) defined in even numbers are deployed as mirrors (see Rules for Mirroring).	"Mirror": "true"	n/a	Both deployments are mirrored.
The CPF merge file to be used to override initial CPF settings for deployed InterSystems IRIS instances (see Deploying with Customized InterSystems IRIS Configurations in the “ICM Reference” chapter).	"UserCPF": "/Samples/cpf/iris.cpf"		Remove unless you are familiar with the CPF merge feature and CPF settings (see the Configuration Parameter File Reference).

Important: The major versions of the image from which you launched ICM and the InterSystems IRIS image you specify using the DockerImage field must match; for example, you cannot deploy a 2019.4 version of InterSystems IRIS using a 2019.3 version of ICM. For information about upgrading ICM before you upgrade your InterSystems containers, see Upgrading ICM Using Distributed Management Mode in the appendix “Sharing Deployments in Distributed Management Mode”.

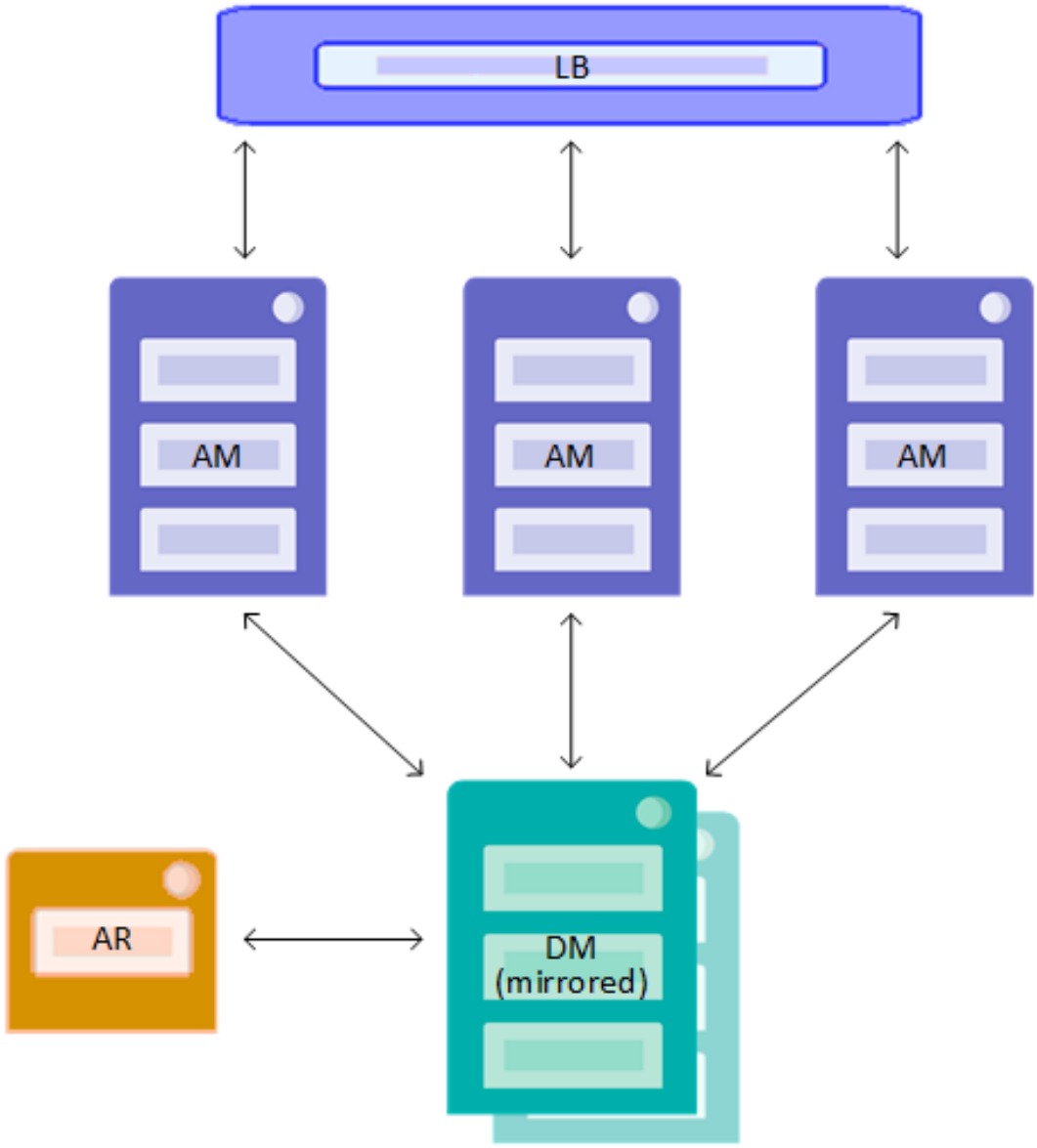
3.4.2 Distributed Cache Cluster Definitions File

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



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

Important: All nodes on which an InterSystems IRIS container is deployed, including DATA, COMPUTE, DM, AM, DS, and QS nodes, require a sharding-enabled InterSystems IRIS license, regardless of the particular configuration involved.

Definition	Field: Value
Two data servers (DM) using a sharding–enabled InterSystems IRIS license, configured as a mirror because “Mirror”: “true” in shared defaults file. Instance type, OS volume size, data volume size override settings in defaults file to meet data server resource requirements.	"Role": "DM", "Count": "2", "LicenseKey": "ubuntu-sharding-iris.key,” "InstanceType": "m4.xlarge", "OSVolumeSize": "32", "DataVolumeSize": "150",
Three application servers (AM) using a sharding–enabled InterSystems IRIS license. Numbering in node names starts at 0003 to follow DM nodes 0001 and 0002. Load balancer for application servers is automatically provisioned.	"Role": "AM", "Count": "3", "LicenseKey": "ubuntu-sharding-iris.key”, "StartCount": "3", "LoadBalancer": "true",
One arbiter (AR) for data server mirror, no license required, use of arbiter image overrides InterSystems IRIS image specified in defaults file. Node is numbered 0006. Instance type overrides defaults file because arbiter requires only limited resources.	"Role": “AR”, "Count": "1", "DockerImage": "intersystems/arbiter:2020.3.0.221.0", "StartCount": "6", "InstanceType": "t2.small",

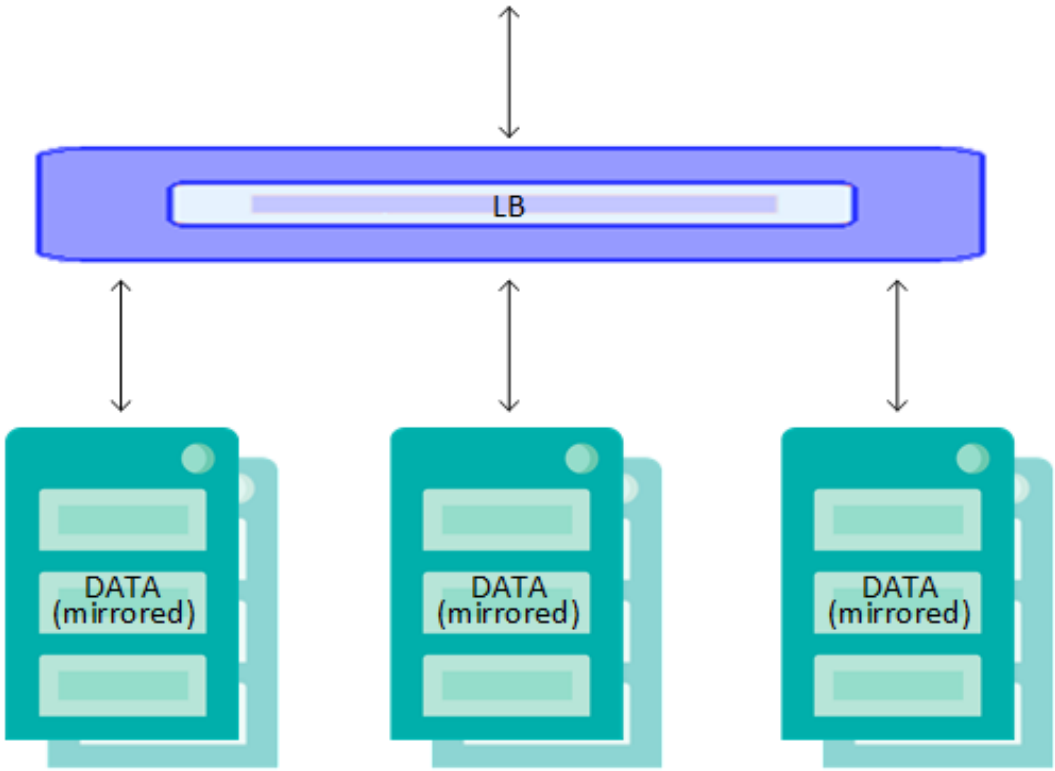
A definitions.json file incorporating the settings in the preceding table would look like this:

```
[
    {
        "Role": "DM",
        "Count": "2",
        "LicenseKey": "standard-iris.key”,
        "InstanceType": "m4.xlarge",
        "OSVolumeSize": "32",
        "DataVolumeSize": "150"
    },
    {
        "Role": "AM",
        "Count": "3",
        "LicenseKey": "standard-iris.key”,
        "StartCount": "3",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "DockerImage": "intersystems/arbiter:2020.3.0.221.0",
        "StartCount": "6",
        "InstanceType": "t2.small"
    }
]
```

3.4.3 Sharded Cluster Definitions File

The definitions.json file for the sharded cluster configuration must define three load-balanced mirrored DATA nodes. This is 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.

Definition	Field: Value
• Six data nodes (DATA) using a sharding-enabled InterSystems IRIS license, configured as three mirrors because “Mirror”: “true” in shared defaults file • Instance type and data volume size override settings in defaults file to meet data node resource requirements • Load balancer for data nodes is automatically provisioned	"Role": "DATA" "Count": "6" "LicenseKey": "ubuntu-sharding-iris.key” "InstanceType": "m4.4xlarge" “DataVolumeSize": "250" "LoadBalancer": "true"
• One arbiter (AR) for data server mirror, no license required, use of arbiter image overrides InterSystems IRIS image specified in defaults file • Node is numbered 0007 • Instance type overrides defaults file because arbiter requires only limited resources	"Role": “AR” "Count": "1" "DockerImage": "intersystems/arbiter:2020.3.0.221.0" "StartCount": "7" "InstanceType": "t2.small"

A definitions.json file incorporating the settings in the preceding table would look like this:

```
[
    {
        "Role": "DATA",
        "Count": "6",
        "LicenseKey": "sharding-iris.key”,
        "InstanceType": "m4.xlarge",
        "DataVolumeSize": "250",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "DockerImage": "intersystems/arbiter:2020.3.0.221.0",
        "StartCount": "7",
        "InstanceType": "t2.small"
    }
]
```

Note: The DATA and COMPUTE node types were added to ICM in Release 2019.3 to support the node-level sharding architecture. Previous versions of this document described the namespace-level sharding architecture, which involves a different, larger set of node types. The namespace-level architecture remains in place as the transparent foundation of the node-level architecture and is fully compatible with it, and the node types used to deploy it are still available in ICM. For information about all available node types, see ICM Node Types.

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 for Data Volume with Sharding” chapter of the *Scalability Guide*.

The recommended best practice is to load-balance application connections across all of the data nodes in a cluster.

3.4.4 Customizing InterSystems IRIS Configurations

Every InterSystems IRIS instance, including those running in the containers deployed by ICM, is installed with a predetermined set of configuration settings, recorded in its configuration parameters file (CPF). You can use the UserCPF field in your defaults file (as illustrated in the /Samples/AWS/defaults.json file in Shared Defaults File) to specify a CPF merge file, allowing you to override one or more of these configuration settings for all of the InterSystems IRIS instances you deploy, or in your definitions file to override different settings for different node types, such as the DM and AM nodes in a distributed cache cluster or the DATA and COMPUTE nodes in a sharded cluster. For example, as described in Planning an InterSystems IRIS Sharded Cluster in the “Sharding” chapter of the *Scalability Guide*, you might want to adjust the size of the database caches on the data nodes in a sharded cluster, which you could do by overriding the value of the **[config]/globals** CPF setting for the DATA definitions only. For information about using a merge file to override initial CPF settings, see Deploying with Customized InterSystems IRIS Configurations in the “ICM Reference” chapter.

A simple CPF merge file is provided in the /Samples/cpf directory in the ICM container, and the sample defaults files in all of the /Samples provider subdirectories include the UserCPF field, pointing to this file. Remove UserCPF from your defaults file unless you are sure you want to merge its contents into the default CPFs of deployed InterSystems IRIS instances.

Information about InterSystems IRIS configuration settings, their effects, and their installed defaults is provided in the Installation Guide, the System Administration Guide, and the Configuration Parameter File Reference.

3.5 Provision the Infrastructure

ICM provisions cloud infrastructure using the HashiCorp Terraform tool.

- The icm provision Command
- Reprovisioning the Infrastructure

- Infrastructure Management Commands

Note: ICM can deploy containers on existing cloud, virtual or physical infrastructure; see Deploying on a Preexisting Cluster for more information.

3.5.1 The icm provision Command

The **icm provision** command allocates and configures host nodes, using the field values provided in the definitions.json and defaults.json files (as well as default values for unspecified parameters where applicable). The input files in the current working directory are used, and during provisioning, the state directory, instances.json file, and log files are created in the same directory (for details on these files see Configuration, State and Log Files in the chapter "Essential ICM Elements").

Because the state directory and the generated instances.json file (which serves as input to subsequent reprovisioning, deployment, and management commands) are unique to a deployment, setting up znd working in a directory for each deployment is generally the simplest effective approach. For example, in the scenario described here, in which two different deployments share a defaults file, the simplest approach would be to set up one deployment in a directory such as /Samples/AWS, as shown in Shared Defaults File, then copy that entire directory (for example to /Samples/AWS2) and replace the first definitions.json file with the second.

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 host nodes and associated components that have been provisioned, and outputs a command line which can be used to unprovision (delete) the infrastructure at a later date.

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

```
$ icm provision
Starting init of ANDY-TEST...
...completed init of ANDY-TEST
Starting plan of ANDY-DM-TEST...
...
Starting refresh of ANDY-TEST...

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

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

Important: Unprovisioning public cloud host nodes in a timely manner avoids unnecessary expense.

Interactions with cloud providers sometimes involve high latency leading to timeouts and internal errors on the provider side, and errors in the configuration files can also cause provisioning to fail. Because the **icm provision** command is fully *reentrant*, it can be issued multiple times until ICM completes all the required tasks for all the specified nodes without error. For more information, see the next section, Reprovisioning the Infrastructure.

3.5.2 Reprovisioning the Infrastructure

To make the provisioning process as flexible and resilient as possible, the **icm provision** command is fully reentrant — it can be issued multiple times for the same deployment. There are two primary reasons for reprovisioning infrastructure by executing the **icm provision** command more than once, as follows:

- Overcoming provisioning errors

  Interactions with cloud providers sometimes involve high latency leading to timeouts and internal errors on the provider side, 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.

- Modifying provisioned infrastructure

  When your needs change, you can modify infrastructure that has already been provisioned, including configurations on which services have been deployed, at any time by changing the characteristics of existing nodes, adding nodes, or removing nodes.

When you repeat the **icm provision** command following an error, if the working directory does not contain the configuration files, you must repeat any location override options, this file does not yet exist, so you must use the **-stateDir** option to specify the incomplete infrastructure you want to continue provisioning. When you repeat the command to modify successfully provisioned infrastructure, however, you do not need to do so; as long as you are working in the directory containing the instances.json file, it is automatically used to identify the infrastructure you are reprovisioning. This is shown in the sections that follow.

3.5.2.1 Overcoming Provisioning Errors

When you issue the **icm provision** command and errors prevent successful provisioning, the state directory is created, but the instances.json file is not. Simply issue the **icm provision** command again, using the**-stateDir** option to specify the state subdirectory's location if it is not in the current working directory. This indicates that provisioning is incomplete and provides the needed information about what has been done and what hasn't. For example, suppose you encounter the problem in the following:

```
$ icm provision
Starting plan of ANDY-DM-TEST...
...completed plan of ANDY-DM-TEST
Starting apply of ANDY-AM-TEST...
Error: Thread exited with value 1
See /Samples/AWS/state/Sample-DS-TEST/terraform.err
```

Review the indicated errors, fix as needed, then run **icm provision** again in the same directory:

```
$ icm provision
Starting plan of ANDY-DM-TEST...
...completed plan of ANDY-DM-TEST
Starting apply of ANDY-DM-TEST...
...completed apply of ANDY-DM-TEST
[...]
To destroy: icm unprovision [-cleanUp] [-force]
```

3.5.2.2 Modifying Provisioned Infrastructure

At any time following successful provisioning — including after successful services deployment using the **icm run** command — you can alter the provisioned infrastructure or configuration by modifying your definitions.json file and executing the

icm provision command again. If changing a deployed configuration, you would then execute the **icm run** command again, as described in Redeploying Services.

You can modify existing infrastructure or a deployed configuration in the following ways.

- To change the characteristics of one or more nodes, change settings within the node definitions in the definitions file. You might want to do this to vertically scale the nodes; for example, in the following definition, you could change the DataVolumeSize setting (see General Parameters) to increase the sizes of the DM nodes' data volumes:

```
{
    "Role": "DM",
    "Count": "2",
    "LicenseKey": "standard-iris.key”,
    "InstanceType": "m4.xlarge",
    "OSVolumeSize": "32",
    "DataVolumeSize": "25"
 },
```

  CAUTION: Modifying attributes of existing nodes such as changing disk sizes, adding CPUs, and so on may cause those nodes (including their persistent storage) to be recreated. This behavior is highly specific to each cloud provider, and caution should be used to avoid the possibility of corrupting or losing data.

  Important: Changes to the Label and Tag fields in the definitions.json file are not supported when reprovisioning.

- To add nodes, modify the definitions.json file in one or more of the following ways:
  - Add a new node type by adding a definition. For example, if you have deployed a sharded cluster with data nodes only, you can add compute nodes by adding an appropriate COMPUTE node definition to the definitions file.
  - Add more of an existing node type by increasing the Count specification in its definition. For example, to add two more application servers to a distributed cache cluster that already has two, you would modify the AM definition by changing "Count": "2" to "Count": "4". When you add nodes to existing infrastructure or a deployed configuration, existing nodes are not restarted or modified, and their persistent storage remains intact.

    Note: When you add data nodes to a deployed sharded cluster after it has been loaded with data, you can automatically redistribute the sharded data across the new servers (although this must be done with the cluster offline); for more information, see Add Shard Data Servers and Rebalance Data in the "Horizontally Scaling InterSystems IRIS for Data Volume with Sharding" chapter of the *Scalability Guide*.

    Generally, there are many application-specific attributes that cannot be modified by ICM and must be modified manually after adding nodes.

  - Add a load balancer by adding "LoadBalancer": "true" to DATA, COMPUTE, AM, or WS node definitions.

- To remove nodes, decrease the Count specification in the node type definition. To remove all nodes of a given type, reduce the count to 0.

  CAUTION: *Do not* remove a definition entirely; Terraform will not detect the change and your infrastructure or deployed configuration will include orphaned nodes that ICM is no longer tracking.

  Important: When removing one or more nodes, you cannot choose which to remove; rather nodes are unprovisioned on a last in, first out basis, so the most recently created node is the first one to be removed. This is especially significant when you have added one or more nodes in a previous reprovisioning operation, as these will be removed before the originally provisioned nodes.

  You can also remove load balancers by removing "LoadBalancer": "true" from a node definition, or changing the value to false.

There are some limitations to modifying an existing configuration through reprovisioning, as follows:

- You cannot remove nodes that store data — DATA, DM, or DS.
- You cannot change a configuration from nonmirrored to mirrored, or the reverse.
- You can add DATA, DS, or DM nodes to a mirrored configuration only in numbers that match the relevant MirrorMap setting, as described in Rules for Mirroring. For example, if the MirrorMap default value of primary,backup is in effect, DATA and DS nodes can be added in even numbers (multiples of two) only to be configured as additional failover pairs, and DM nodes cannot be added; if MirrorMap is primary,backup,async, DATA or DS nodes can be added in multiples of three to be configured as additional three-member mirrors, or as a pair to be configured as an additional mirror with no async, and a DM node can be added only if the existing DM mirror does not currently include an async.
- You can add an arbiter (AR node) to a mirrored configuration, but it must be manually configured as arbiter, using the management portal or **^MIRROR** routine, for each mirror in the configuration.

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

Remember that after reprovisioning a deployed configuration, you must issue the **icm run** command again to redeploy.

3.5.3 Infrastructure Management Commands

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

Many ICM command options can be used with more than one command. For example, the **-role** option can be used with a number of commands to specify the of the nodes for which the command should be run — for example, **icm inventory -role AM** lists only the nodes in the deployment that are of type AM — and the **-image** option, which specifies an image from which to deploy containers for both the icm run and icm upgrade commands. For complete lists of ICM commands and their options, see ICM Commands and Options in the "ICM Reference" chapter.

3.5.3.1 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                                                  Region    Zone
-------            ----------     --------                                                  ------    ----
ANDY-DM-TEST-0001+ 00.53.183.209  ec2-00-53-183-209.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-DM-TEST-0002- 00.53.183.185  ec2-00-53-183-185.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-AM-TEST-0003  00.56.59.42    ec2-00-56-59-42.us-west-1.compute.amazonaws.com    us-west-1 c
ANDY-AM-TEST-0005  00.67.1.11     ec2-00-67-1-11.us-west-1.compute.amazonaws.com     us-west-1 c
ANDY-AM-TEST-0003  00.193.117.217 ec2-00-193-117-217.us-west-1.compute.amazonaws.com us-west-1 c
ANDY-LB-TEST-0002  (virtual AM)   ANDY-AM-TEST-1546467861.amazonaws.com              us-west-1 c
ANDY-AR-TEST-0006  00.53.201.194  ec2-00-53-201-194.us-west-1.compute.amazonaws.com  us-west-1 c
```

Note: When mirrored nodes are part of a configuration, initial mirror failover assignments are indicated by a **+** (plus) following the machine name of each intended primary and a **-** (minus) following the machine name of each intended backup, as shown in the preceding example. These assignments can change, however; following deployment, use the **icm ps** command to display the mirror member status of the deployed nodes.

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                                                  Region    Zone
-------           ----------     --------                                                  ------    ----
ANDY-AM-TEST-0003 00.56.59.42    ec2-00-56-59-42.us-west-1.compute.amazonaws.com    us-west-1 c
ANDY-AM-TEST-0005 00.67.1.11     ec2-00-67-1-11.us-west-1.compute.amazonaws.com     us-west-1 c
ANDY-AM-TEST-0003 00.193.117.217 ec2-00-193-117-217.us-west-1.compute.amazonaws.com us-west-1 c
```

If the fully qualified DNS names from the cloud provider are too wide for a readable display, you can use the **-options** option with the **wide** argument to make the output wider, for example:

```
icm inventory -options wide
```

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

3.5.3.2 icm ssh

The **icm ssh** command runs an arbitrary command on the specified host 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 ANDY-DM-TEST-0001...
Executing command 'ping -c 5 intersystems.com' on ANDY-DM-TEST-0002...
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/ssh.out
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0002/ssh.out
```

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

```
$ icm ssh -command "df -k" -machine ANDY-DM-TEST-0001
Executing command 'df -k' on ANDY-DM-TEST-0001...
...output in ./state/ANDY-DM-TEST/ANDY-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 host 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. The icm scp command, which securely copies a file or directory from the local ICM container to the host OS of the specified node or nodes, is frequently used with **icm ssh**.

3.5.3.3 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
- Redeploying Services
- Container Management Commands
- Service Management Commands

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.

  The InterSystems IRIS images distributed by InterSystems, and how to use one as a base for a custom image that includes your InterSystems IRIS-based application, are described in detail in Running InterSystems Products in Containers. In addition to the **`iris`** image for standard platforms, InterSystems IRIS images for ARM platforms are named **`iris-arm64`**.

  When deploying the **`iris`** image, you can override one or more InterSystems IRIS configuration settings for all of the **`iris`** containers you deploy, or override different settings for containers deployed on different node types; for more information, see Deploying with Customized InterSystems IRIS Configurations in the "ICM Reference" chapter.

- **`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 DATA or DATA and COMPUTE nodes in a node-level sharded cluster, AM or DM nodes in other configurations.

- **`iam`** — Contains the InterSystems API Manager; for more information, see Deploying InterSystems API Manager in the "ICM Reference" chapter.

- **`sam`** — Contains the SAM Manager component of the System Alerting and Monitoring (SAM) cluster monitoring solution; for more information, see Monitoring in ICM in the "ICM Reference" chapter.

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. For a list of node types and corresponding InterSystems images, see ICM Node Types in the "ICM Reference" chapter.

Important: If the wrong InterSystems image is specified for a node by the DockerImage field or the **-image** option of the **icm run** command — for example, if the **`iris`** image is specified for an AR (arbiter) node, or any InterSystems image for a CN node — deployment fails, with an appropriate message from ICM. Therefore, when the DockerImage field specifies the **`iris`** 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).

The major versions of the image from which you launched ICM and the InterSystems images you specify using the DockerImage field must match; for example, you cannot deploy a 2019.4 version of InterSystems IRIS using a 2019.3 version of ICM. For information about upgrading ICM before you upgrade your InterSystems containers, see Upgrading ICM Using Distributed Management Mode in the appendix "Sharing Deployments in Distributed Management Mode".

For information about using InterSystems images other than **`iris`** (such as the **`arbiter`** image) outside of ICM, contact the InterSystems Worldwide Response Center (WRC).

Note: Container images from InterSystems comply with the Open Container Initiative (OCI) specification, and are built using the Docker Enterprise Edition engine, which fully supports the OCI standard and allows for the images to be certified and featured in the Docker Hub registry.

InterSystems images are built and tested using the widely popular container Ubuntu operating system, and are therefore supported on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.

For a brief introduction to the use of InterSystems IRIS in containers, including a hands-on experience, see *First Look: InterSystems Products in Containers*; for detailed information about deploying InterSystems IRIS and InterSystems IRIS-based applications in containers using methods other than ICM, see *Running InterSystems Products in Containers*.

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 does not transmit the InterSystems IRIS password (however specified) in plain text, but instead uses a cryptographic hash function to generate a hashed password and salt locally, then sends these using SSH to the deployed InterSystems IRIS containers on the host 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:2020.3.0.221.0"
```

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 mirrored DATA nodes and an AR (arbiter) node:

  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 and to enable mirroring (as described in Rules for Mirroring):

     ```
     "DockerImage": "intersystems/iris:2020.3.0.221.0",
     "Mirror": "true"
     ```

  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:2020.3.0.221.0"
      }
     ```

  3. 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 DATA nodes; a container named **iris** containing an ISCAgent to act as mirror arbiter is deployed on the AR node; ICM performs the needed sharding 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-arm64:2020.3.0.221.0"
      },
      {
          "Role": "WS",
          "Count": "3",
          "DockerImage": "intersystems/webgateway:2020.3.0.221.0"
      }
     ```

  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 another **icm run** command to deploy the custom container on the DM node, for example either of the following:

```
icm run -container customsensors -image myrepo/sensors:1.0 -role DM
```

```
icm run -container customsensors -image myrepo/sensors:1.0 -machine ANDY-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 ANDY-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 registry 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 IRISCLUSTER 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 name you specified, and this becomes the default 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:2020.3.0.221.0
```

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

```
$ icm run
Executing command 'docker login' on ANDY-DATA-TEST-0001...
...output in /Samples/AWS/state/ANDY-DATA-TEST/ANDY-DATA-TEST-0001/docker.out
...
Pulling image intersystems/iris:2020.3.0.221.0 on ANDY-DATA-TEST-0001...
...pulled ANDY-DATA-TEST-0001 image intersystems/iris:2020.3.0.221.0
...
Creating container iris on ANDY-DATA-TEST-0002...
...
Copying license directory /Samples/license/ to ANDY-DATA-TEST-0003...
...
Starting container iris on ANDY-DATA-TEST-0004...
...
Waiting for InterSystems IRIS to start on ANDY-DATA-TEST-0002...
...
Configuring SSL on ANDY-DATA-TEST-0001...
...
Enabling ECP on ANDY-DATA-TEST-0003...
...
Setting System Mode on ANDY-DATA-TEST-0002...
...
Acquiring license on ANDY-DATA-TEST-0002...
...
Enabling shard service on ANDY-DATA-TEST-0001...
```

```
...
Assigning shards on ANDY-DATA-TEST-0001...
...
Configuring application server on ANDY-DATA-TEST-0003...
...
Management Portal available at:
http://ec2-00-56-140-23.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. When multiple data nodes are deployed, the provided Management Portal link is for the lowest numbered among them, in this case the instance, running in the IRIS container on ANDY-DATA-TEST-001.

3.6.2 Redeploying Services

To make the deployment process as flexible and resilient as possible, the **icm run** command is fully reentrant — it can be issued multiple times for the same deployment. When an **icm run** command is repeated, ICM stops and removes the affected containers (the equivalent of **icm stop** and **icm rm**), then creates and starts them from the applicable images again, while preserving the storage volumes for InterSystems IRIS instance-specific data that it created and mounted as part of the initial deployment pass (see Storage Volumes Mounted by ICM in the "ICM Reference" chapter).

There are four primary reasons for redeploying services by executing an **icm run** command more than once, as follows:

- Redeploying the existing containers with their existing storage volumes.

  To replace deployed containers with new versions while preserving the instance-specific storage volumes of the affected InterSystems IRIS containers, thereby redeploying the existing instances, simply repeat the original **icm run** command that first deployed the containers. You might do this if you have made a change in the definitions files that requires redeployment, for example you have updated the licenses in the directory specified by the LicenseDir field.

- Redeploying the InterSystems IRIS containers without the existing storage volumes.

  To replace the InterSystems IRIS containers in the deployment without preserving their instance-specific storage volumes, you can delete that data for those instances before redeploying using the following command:

  ```
  icm ssh -command "sudo rm -rf /<mount_dir>/*/*"
  ```

  where *mount_dir* is the directory (or directories) under which the InterSystems IRIS data, WIJ, and journal directories are mounted (which is /irissys/ by default, or as configured by the DataMountPoint, WIJMountPoint, Journal1MountPoint, and Journal2MountPoint fields; for more information, see Storage Volumes Mounted by ICM in the "ICM Reference" chapter). You can use the **-role** or **-machine** options to limit this command to specific nodes, if you wish. When you then repeat the **icm run** command that originally deployed the InterSystems IRIS containers, those that still have instance-specific volumes are redeployed as the same instances, while those for which you deleted the volumes are redeployed as new instances.

- Deploying services on nodes you have added to the infrastructure, as described in Reprovisioning the Infrastructure.

  When you repeat an **icm run** command after adding nodes to the infrastructure, containers on the existing nodes are redeployed as described in the preceding (with their storage volumes, or without if you have deleted them) while new containers are deployed on the new nodes. This allows the existing nodes to be reconfigured for the new deployment topology, if necessary.

- Overcoming deployment errors.

  If the **icm run** command fails on one or more nodes due to factors outside ICM's control, such as network latency and disconnects or interruptions in cloud provider service (as indicated by error log messages), you can issue the command again; in most cases, deployment will succeed on repeated tries. If the error persists, however, and requires manual intervention — for example, if it is caused by an error in one of the configuration files — you may need to delete the storage volumes on the node or nodes affected, as described in the preceding, before reissuing **icm run** after fixing the problem. This is because ICM recognizes a node without instance-specific data as a new node, and marks the storage volumes of an InterSystems IRIS container as fully deployed only when all configuration is successfully completed;

if configuration begins but fails short of success and the volumes are not marked, ICM cannot redeploy on that node. In a new deployment, you may find it easiest to issue the command **icm ssh -command "sudo rm -rf /irissys/*/*"** without **-role** or **-machine** constraints to roll back all nodes on which InterSystems IRIS is to be redeployed.

3.6.3 Container Management Commands

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

Many ICM command options can be used with more than one command. For example, the **-role** option can be used with a number of commands to specify the type of node for which the command should be run — for example, icm inventory **-role AM** lists only the nodes in the deployment that are of type AM — and the **-image** option, which specifies an image from which to deploy containers for both the **icm run** and icm upgrade commands. For complete lists of ICM commands and their options, see ICM Commands and Options in the "ICM Reference" chapter.

3.6.3.1 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
-------             ----------    --------- ------ ------  -----
ANDY-DATA-TEST-0001 00.56.140.23  iris      Up     healthy intersystems/iris:2020.3.0.221.0
ANDY-DATA-TEST-0002 00.53.190.37  iris      Up     healthy intersystems/iris:2020.3.0.221.0
ANDY-DATA-TEST-0003 00.67.116.202 iris      Up     healthy intersystems/iris:2020.3.0.221.0
ANDY-DATA-TEST-0004 00.153.49.109 iris      Up     healthy intersystems/iris:2020.3.0.221.0
```

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
  -------             ----------    --------- ------ ------  ------  -----
  ANDY-DATA-TEST-0001 00.56.140.23  iris      Up     healthy PRIMARY intersystems/iris:2020.3.0.221.0
  ANDY-DATA-TEST-0002 00.53.190.37  iris      Up     healthy BACKUP  intersystems/iris:2020.3.0.221.0
  ANDY-DATA-TEST-0003 00.67.116.202 iris      Up     healthy PRIMARY intersystems/iris:2020.3.0.221.0
  ANDY-DATA-TEST-0004 00.153.49.109 iris      Up     healthy BACKUP  intersystems/iris:2020.3.0.221.0
  ```

  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.

3.6.3.2 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 ANDY-DATA-TEST-0001...
Stopping container iris on ANDY-DATA-TEST-0002...
Stopping container iris on ANDY-DATA-TEST-0004...
Stopping container iris on ANDY-DATA-TEST-0003...
...completed stop of container iris on ANDY-DATA-TEST-0004
...completed stop of container iris on ANDY-DATA-TEST-0001
...completed stop of container iris on ANDY-DATA-TEST-0002
...completed stop of container iris on ANDY-DATA-TEST-0003
```

3.6.3.3 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 ANDY-DATA-TEST-0002...
Starting container iris on ANDY-DATA-TEST-0002...
...completed start of container iris on ANDY-DATA-0002
```

3.6.3.4 icm pull

The **icm pull** command downloads the specified image to the specified machines. For example, to add an image to data node 1 in the sharded cluster:

```
$ icm pull -image intersystems/webgateway:2020.3.0.221.0 -role DATA
Pulling ANDY-DATA-TEST-0001 image intersystems/webgateway:2020.3.0.221.0...
...pulled ANDY-DATA-TEST-0001 image intersystems/webgateway:2020.3.0.221.0
```

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-arm64:2020.3.0.221.0",
```

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.

3.6.3.5 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.

3.6.3.6 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 an application server:

```
$ icm upgrade -image intersystems/iris:2020.3.0.343.0 -machine ANDY-AM-TEST-0003
Pulling ANDY-AM-TEST-0003 image intersystems/iris:2020.3.0.343.0...
...pulled ANDY-AM-TEST-0003 image intersystems/iris:2020.3.0.343.0
Stopping container ANDY-AM-TEST-0003...
...completed stop of container ANDY-AM-TEST-0003
Removing container ANDY-AM-TEST-0003...
...removed container ANDY-AM-TEST-0003
Running image intersystems/iris:2020.3.0.343.0 in container ANDY-AM-TEST-0003...
...running image intersystems/iris:2020.3.0.343.0 in container ANDY-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.

Note: The major versions of the image from which you launch ICM and the InterSystems images you deploy must match. For example, you cannot deploy a 2019.4 version of InterSystems IRIS using a 2019.3 version of ICM.

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

For important information about upgrading InterSystems IRIS containers, see Upgrading InterSystems IRIS Containers in *Running InterSystems Products in Containers*.

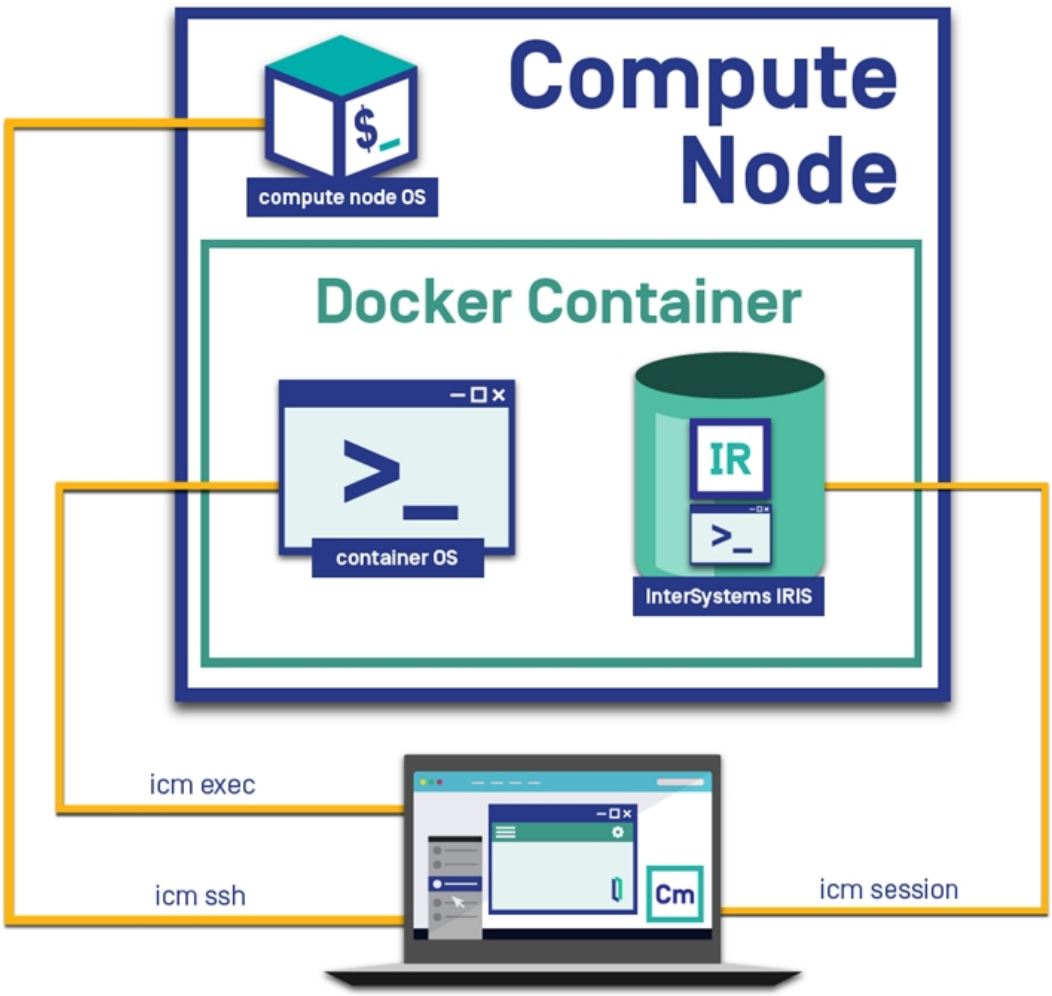
3.6.4 Service Management Commands

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

Many ICM command options can be used with more than one command. For example, the **-role** option can be used with a number of commands to specify the type of node for which the command should be run — for example, **icm exec -role AM** runs the specified command on only the nodes in the deployment that are of type AM — and the **-image** option, which specifies an image from which to deploy containers for both the icm run and icm upgrade commands. For complete lists of ICM commands and their options, see ICM Commands and Options in the "ICM Reference" chapter.

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 host 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



3.6.4.1 icm exec

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

```
$ icm exec -command "df -k" -machine ANDY-DM-TEST-0001
Executing command in container iris on ANDY-DM-TEST-0001
...output in ./state/ANDY-DM-TEST/ANDY-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 ANDY-AM-TEST-0004 -interactive
Executing command 'bash' in container iris on ANDY-AM-TEST-0004...
[root@localhost /] $ whoami
root
[root@localhost /] $ hostname
iris-ANDY-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.

The **icm cp** command, which copies a local file or directory on the specified node into the specified container, is useful with **icm exec**.

3.6.4.2 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; the default is the ICM-created namespace (IRISCLUSTER by default). For example:

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

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

%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 ANDY-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 ANDY-AM-TEST-0003...
Executing command in container iris on ANDY-AM-TEST-0004...
Executing command in container iris on ANDY-AM-TEST-0005...
...output in ./state/ANDY-AM-TEST/ANDY-AM-TEST-0003/ssh.out
...output in ./state/ANDY-AM-TEST/ANDY-AM-TEST-0004/ssh.out
...output in ./state/ANDY-AM-TEST/ANDY-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 ANDY-DM-TEST-0001
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/docker.out

0
```

The icm sql command, which runs an arbitrary SQL command against the containerized InterSystems IRIS instance on the specified node (or all nodes), is similar to **icm session**.

3.6.4.3 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.

3.6.4.4 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 ANDY-DM-TEST-0001...
...output in ./state/ANDY-DM-TEST/ANDY-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; the default is the ICM-created namespace (IRISCLUSTER by default).

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.

The **icm sql** command can also be interactively on a single node, opening an InterSystems IRIS SQL Shell (see the "Using the SQL Shell Interface" chapter of *Using InterSystems SQL*). 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. For example:

```
$ icm sql -interactive -machine ANDY-QS-TEST-0002
SQL Command Line Shell
----------------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter <command>, 'q' to quit, '?' for help.
```

As with the noninteractive command, you can use the **-namespace** option interactively to specify the namespace in which the SQL shell runs; the default is the ICM-created namespace (IRISCLUSTER by default).

3.6.4.5 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 ANDY-DM-TEST-0002
Executing command 'status --no-stream' on ANDY-DM-TEST-0002...
...output in ./state/ANDY-DM-TEST/ANDY-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. If the state subdirectory is not in the current working directory, the **-stateDir** option is required to specify its location. 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 -cleanUp
Type "yes" to confirm: yes
Starting destroy of ANDY-DM-TEST...
Starting destroy of ANDY-AM-TEST...
Starting destroy of ANDY-AR-TEST...
...completed destroy of ANDY-AR-TEST
...completed destroy of ANDY-AM-TEST
...completed destroy of ANDY-DM-TEST
Starting destroy of ANDY-TEST...
...completed destroy of ANDY-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

This chapter provides detailed information about various aspects of ICM and its uses.

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.

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
provision	Provisions host nodes	n/a
inventory	Lists provisioned host nodes	-machine, -role, -json, -options
unprovision	Destroys host nodes	-stateDir, -cleanup, -force
merge	Merges infrastructure provisioned in separate regions or provider platforms into a new definitions file for multiregion or multiprovider deployment	-options, -localPath
ssh	Executes an operating system command on one or more host nodes	-command, -machine, -role
scp	Copies a local file to one or more host nodes	-localPath, -remotePath, -machine, -role

Command	Description	Important Options
run	Deploys a container on host nodes	-image, -container, -namespace, -options, -iscPassword, -command, -machine, -role
ps	Displays run states of containers deployed on host nodes	-container, -json
stop	Stops containers on one or more host nodes	-container, -machine, -role
start	Starts containers on one or more host nodes	-container, -machine, -role
pull	Downloads an image to one or more host nodes	-image, -container, -machine, -role
rm	Deletes containers from one or more host nodes	-container, -machine, -role
upgrade	Replaces containers on one or more host 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 InterSystems 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
install	Installs InterSystem IRIS instances from a kit in containerless mode	-machine, -role
uninstall	Uninstalls InterSystems IRIS instances installed from a kit in containerless mode	-machine, -role
docker	Executes a Docker command on one or more host nodes	-container, -machine, -role

Table 4–2: ICM Command-Line Options

Option	Description	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)
-force	Don't confirm before unprovisioning	false	Unprovision the Infrastructure

Option	Description	Default	Described in
-cleanUp	Delete state directory after unprovisioning	false	
-machine *regexp*	Machine name pattern match used to specify the node or nodes for which the command is run	(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 DATA or AM	(all)	
-namespace *namespace*	Namespace to create on deployed InterSystems IRIS instances and set as default execution namespace for the **session** and **sql** commands	IRISCLUSTER	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
-options *options*	Additional Docker options	none	icm inventory icm run icm exec icm session Deploying Across Multiple Regions or Providers Using ICM with Custom and Third-Party Containers
-container *name*	Name of the container	**icm ps** command: (all) other commands: iris	icm run icm ps
-command *cmd*	Command or query to execute	none	icm ssh icm run icm exec icm session icm sql

Option	Description	Default	Described in
-interactive	Redirect input/output to console for the **exec** and **ssh** commands	false	icm ssh icm exec icm sql
-localPath *path*	Local file or directory	none	icm scp icm cp Deploying Across Multiple Regions or Providers
-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 Configuration Parameters

These tables describe the fields you can include in the configuration files (see Configuration, State and Log Files in the "Essential ICM Elements" chapter and Define the Deployment in the "Using ICM chapter") to provide ICM with the information it needs to execute provisioning and deployment tasks and management commands. To look up a parameter by name, use the alphabetical list, which includes links to the tables containing the parameter definitions.

- General Parameters
- Security-Related Parameters
- Port and Protocol Parameters
- CPF Parameters
- Provider-Specific Parameters
  - Amazon Web Services (AWS)
  - Google Cloud Platform (GCP)
  - Microsoft Azure (Azure)
  - Tencent Cloud (Tencent)
  - VMware vSphere (vSphere)
- Device Name Parameters
- Alphabetical List of User Parameters

4.2.1 General Parameters

The fields in the following table are all used with all cloud providers, and some are used with vSphere and Preexisting as well.

The two rightmost columns indicate whether each parameter is required in every deployment or optional, and whether it must be included (when used) in either defaults.json or definitions.json, is recommended for one file or the other, or can be used in either. For example,

- A single deployment is always on a single selected provisioning platform (even if subsequently merged with another to create a multiprovider deployment), therefore the Provider parameter is required and must be in the defaults file.
- Each node type must be specified but a deployment can include multiple node types, thus the Role parameter is required in each definition in the definitions file.
- Because each node that runs InterSystems IRIS must have a license, but other nodes don't need one, the LicenseKey setting is required and generally appears in the appropriate definitions in the definitions file.
- At least one container must be deployed on each node in the deployment, but a single container may be deployed on all the nodes (for instance **`iris`**/**`iris-arm64`** across a sharded cluster consisting of DATA nodes only) or different containers on different node types (**`iris`**/**`iris-arm64`** on DM and AM, **`webgateway`** on WS, **`arbiter`** on AR in a distributed cache cluster). For this reason the DockerImage parameter is required and can appear in the defaults file, the definitions file, or both (to specify a default image but override it for one or more node types).
- Like the image to be deployed, the size of the OS volume can be specified for all nodes in the defaults file, for one or more node types in the definitions file, or in both, but because it has a default it is optional.

Note: If no default is listed for a parameter, it does not have one.

Provider	Definition	Use is ...	Config file
Provider	Platform to provision infrastructure on; see Provisioning Platforms.	required	defaults
Label Tag	Fields in naming scheme for provisioned cloud nodes: *Label-Role-Tag-NNNN*, for example **ANDY-DATA-TEST-0001**; should indicate ownership and purpose, to avoid conflicting with others. Multiple deployments should not share the same Label and Tag. Cannot contain dashes.	required	defaults
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.	required	defaults
LicenseKey	Sharding-enabled license key for the InterSystems IRIS instance on one or more provisioned DATA, COMPUTE, DM, AM, DS, or QS nodes, staged within the ICM container in the location specified by the LicenseDir field (above).	required	definitions recommended

Provider	Definition	Use is ...	Config file
Region (Azure equivalent: Location)	Geographical region of provider's compute resources in which infrastructure is provisioned. For information on deploying a single configuration in more than one region, see Deploying Across Multiple Regions or Providers. Provider-specific information, including provider documentation: • AWS — Example: us-west-1; see About AWS Regions and Availability Zones. • GCP — Example: us-east1; see Regions and Zones. Can be a comma-separated list for mutliregion deployment. • Azure — Example: Central US; see Azure geographies. (Use Location instead of Region.) • Tencent — Example: na-siliconvalley (West US/Silicon Valley); see Regions and Availability Zones.	required	defaults
Zone	Availability zone within the specified region (see above) in which to locate a node or nodes to be provisioned. For information on deploying a single configuration in more than one zone, see Deploying Across Multiple Zones. Provider-specific information: • AWS — Example: us-west-1c in region us-west-1; see Regions and Zones. • GCP — Example: us-east1-b in region us-east1; see Regions and Zones. • Azure — Example: 1 in region Central US; see What are Availability Zones in Azure?. **Note:** Some Azure regions do not support availability zones. To deploy to such a regionset Zone to the empty string or omit it altogether: • Tencent — Example: na-siliconvalley-1 in region na-siliconvalley; see Regions and Availability Zones.	required	defaults
ZoneMap	When deploying across multiple zones (see Deploying Across Multiple Zones), specifies which nodes are deployed in which zones. Default: 0,1,2,...,255.	optional	definitions
Mirror	If true, InterSystems IRIS instances on DATA, DM, and DS nodes are deployed as mirrors; see Mirrored Configuration Requirements. Default: false.	optional	defaults

Provider	Definition	Use is ...	Config file
MirrorMap	Determines mirror member types of mirrored DATA, DS, and DM nodes, enabling deployment of DR async mirror members; see Rules for Mirroring. Default: primary,backup; the term async can be added one or more times to this, for example primary,backupo,async,async.	optional	definitions
ISCPassword	Password that will be set for the predefined user accounts on the InterSystems IRIS instances on one or more provisioned nodes. Corresponding command-line option: **-iscPassword**. If both parameter and option are omitted, ICM prompts for the password. For more information see The icm run Command.	optional	defaults
Namespace	Namespace to be created on deployed InterSystems IRIS instances. This namespace is the default namespace for the icm session and icm sql commands, and can also be specified or overridden by the command-line option **-namespace**. Default: IRISCLUSTER.	optional	defaults
DockerImage	Docker image to be used for in deployment by icm run command. Must include the repository name (see Repositories in the Docker documentation). Can be specified for all nodes in defaults.json and optionally overridden for specific node definitions in definitions.json. Can also be specified or overridden using the command-line option **-image**.	required	
DockerRegistry	DNS name of the server hosting the Docker repository storing the image specified by DockerImage (see About Registry in the Docker documentation). If not included, ICM uses Docker's public registry at docker.com. For information about the InterSystems Container Registry (ICR), see Downloading the ICM Image in the "Using ICM" chapter.	required	defaults
DockerUsername	Username to use along with DockerPassword (below) for logging into the Docker repository specified in DockerImage (above) on the registry specified by DockerRegistry (above). Not required for public repositories. If not included and the repository specified by DockerImage is private, login fails.	required	defaults
DockerPassword	Password to use along with DockerUsername (above) for logging into the Docker registry. 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. (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**.)	required	defaults

Provider	Definition	Use is ...	Config file
DockerVersion	Version of Docker installed on provisioned nodes. The version in each /Samples/.../defaults.json is generally correct for the platform; however. if your organization uses a different version of Docker, you may want that version installed on the nodes instead. **Important:** Container images from InterSystems comply with the Open Container Initiative (OCI) specification, and are built using the Docker Enterprise Edition engine, which fully supports the OCI standard and allows for the images to be certified and featured in the Docker Hub registry. InterSystems images are built and tested using the widely popular container Ubuntu operating system, and are therefore supported on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.	optional	defaults
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. For more information about Docker EE see Docker Enterprise in the Docker documentation.	optional	defaults
Overlay	Determines the Docker overlay network type; normally "weave", but may be set to "host" for development, performance, or debug purposes, or when deploying on a preexisting cluster. Default: weave (host when deploying on a preexisting cluster). For more information see Use overlay networks in the Docker documentation and How the Weave Net Docker Network Plugins Work in the Weave documentation.	optional	defaults
DockerStorageDriver	Determines the storage driver used by Docker (see Docker storage drivers in the Docker documentation). Values include overlay2 (the default) and btrfs. If set to overlay2, FileSystem (see below) must be set to xfs; if set to btrfs, FileSystem must be set to btrfs..	optional	defaults
FileSystem	Type of file system to use for persistent volumes on provisioned nodes. Valid values are xfs and btrfs. Default: xfs. If DockerStorageDriver (above) is set to overlay2, FileSystem must be set to xfs; if DockerStorageDriver is btrfs, FileSystem must be btrfs.	optional	defaults recommended

Provider	Definition	Use is ...	Config file
OSVolumeSize	Size (in GB) of the OS volume for a node or nodes in the deployment. Default: 32. May be limited by or ignored in favor of settings specific to the applicable parameters specifying machine image or template, instance type, or OS volume type parameters (see Provider-Specific Parameters).	optional	
DataVolumeSize WIJVolumeSize Journal1VolumeSize Journal2VolumeSize	Size (in GB) of the corresponding persistent storage volume to create for **`iris`** containers. For example, DataVolumeSize determines the size of the data volume. Default: 10, although DataVolumeSize must be at least 60 for Tencent deployments. May be limited by the applicable volume type parameter (see Provider-Specific Parameters). Each volume also has a corresponding device name parameter (for example, DataDeviceName; see Device Name Parameters) and mount point parameter (for example, DataMountPoint; see immediately below and Storage Volumes Mounted by ICM).	optional	
DataMountPoint WIJMountPoint Journal1MountPoint Journal2MountPoint	The location within **`iris`** containers at which the corresponding persistent volume is mounted. For example, DataMountPoint determines the location for the data volume. For more information, see Storage Volumes Mounted by ICM. Defaults: /irissys/{ data | wij | journal1j | journal2j }. Each volume also has a corresponding device name parameter (for example, DataDeviceName; see Device Name Parameters) and size parameter (for example, DataVolumeSize; see above).	optional	
Containerless	If true, enables containerless mode, in which InterSystems IRIS is deployed from an installation kit rather than a container; see the appendix Containerless Deployment. Default: false.	optional	defaults
Role	Role of the node or nodes to be provisioned by a given entry in the definitions file, for example DM or DATA; see ICM Node Types.	required	definitions
Count	Number of nodes to provision from a given entry in the definitions file. Default: 1.	required	definitions
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*-0003.	optional	definitions
LoadBalancer	If true in definitions of node type DATA, COMPUTE, AM, or WS, a predefined load balancer is automatically provisioned on providers AWS, GCP, Azure, and Tencent (see Predefined Load Balancer). If true in definitions of node type CN or VM, a generic load balancer is added if other parameters are included in the definition (see Generic Load Balancer). Default: false.	optional	definitions

Provider	Definition	Use is ...	Config file
AlternativeServers	Remote server selection algorithm for definitions of type WS (see Node Type: Web Server). Valid values are LoadBalancing and FailOver. Default: LoadBalancing.	optional	definitions
ApplicationPath	Application path to create for definitions of type WS. Do not include a trailing slash.	optional	definitions
IAMImage	InterSystems API Manager (IAM) image; no default.	optional	definitions
PostgresImage	Postgres image (optional IAM component); default: postgres:11.6.	optional	definitions
PrometheusImage	Prometheus image (System Alerting and Monitoring [SAM] component); default: prom/prometheus:v2.17.1.	optional	definitions
AlertmanagerImage	Alertmanager image (SAM component); default: prom/alertmanager:v0.20.0.	optional	definitions
GrafanaImage	Grafana image (SAM component); default: grafana/grafana:6.7.1.	optional	definitions
NginxImage	Nginx image (SAM component); default: nginx:1.17.9-alpine.	optional	definitions
UserCPF	Merge file to be used to customize the CPFs InterSystems IRIS instances during deployment (see Deploying with Customized InterSystems IRIS Configurations).	optional	
SystemMode	String to be shown in the masthead of the Management Portal of the InterSystems IRIS instances on one or more provisioned nodes. Certain values (LIVE, TEST, FAILOVER, DEVELOPMENT) trigger additional changes in appearance. Default: blank. This setting can also be specified by adding [Startup]/SystemMode to the CPF merge file (see previous entry).	optional	

4.2.2 Security-related Parameters

The parameters in the following table are used to provide access and identify required files and information so that ICM can communicate securely with the provisioned nodes and deployed containers. They are all required, in the defaults file only.

- For information about using scripts provided with ICM to generate these files, see Obtain Security-Related Files in the "Using ICM" chapter.
- For information about how ICM uses the security files you provide to communicate securely with provisioned nodes and services on them, see ICM Security in this chapter
- For general information about using the SSH protocol, see SSH PROTOCOL from SSH Communications Security.
- For information about using TLS with Docker, see Protect the Docker daemon socket in the Docker documentation.

- For general information about using TLS with InterSystems IRIS, see Using 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 TLS to secure connections between mirror members, see Securing Mirror Communication with TLS Security in the *High Availability Guide*.

Parameter	Definition

Parameter	Definition
Provider-specific credentials and account parameters; to see detailed instructions for obtaining the files and values, click the provider link	• AWS Credentials: Path to a file containing the public/private keypair for an AWS account. • GCP Credentials: Path to a JSON file containing the service account key for a GCP account. Project: GCP project ID. • Azure SubscriptionId: A unique alphanumeric string that identifies a Microsoft Azure subscription. TenantId: A unique alphanumeric string that identifies the Azure Active Directory directory in which an application was created. ClientId, ClientSecret: Credentials identifying and providing access to an Azure application. • Tencent SecretID, SecretKey: Unique alphanumeric strings that identify and provide access to a Tencent Cloud account. • vSphere VSphereUser, VSpherePassword: Credentials for vSphere operations.
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 (see AMI parameter in AWS Parameters); usually ec2-user for Red Hat Enterprise Linux instances and ubuntu for Ubuntu images • GCP — At user's discretion • Azure — At user's discretion • Tencent — As per image (see ImageId parameter in Tencent Parameters) • vSphere — As per VM template (see Template parameter in vSphere Parameters) • Preexisting — See SSH in the appendix "Deploying on a Preexisting Cluster"
SSHPassword	Initial password for the user specified by SSHUser. Required for marketplace Docker images and deployments of type vSphere, Azure, and PreExisting. This password is used only during provisioning, at the conclusion of which password logins are disabled.
SSHOnly	If true, ICM does not attempt SSH password logins during provisioning, for providers vSphere and PreExisting only. Because this prevents ICM from logging in using a password, it requires that you stage your public SSH key (as specified by the SSHPublicKey field, below) on each node. Default: false.

Parameter	Definition

SSHPublicKey	Path within the ICM container of the public key of the 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	Path within the ICM container of the private key of the SSH public private key pair; required for all deployments in RSA format, for example:
	`-----BEGIN RSA PRIVATE KEY-----`
	`MIIEogIBAAKCAQEAoa0ex+JKzC2Nka1`
	`-----END RSA PRIVATE KEY-----`

TLSKeyDir	Directory within the ICM container 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
SSLConfig	Path within the ICM container to an 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).
PrivateSubnet	If true, ICM deploys on an existing private subnet, or creates and deploys on a new private subnet, for use with a bastion host; see Deploying on a Private Network.
net_vpc_cidr	CIDR of the existing private network to deploy on; see Deploy Within an Existing Private Network.
net_subnet_cidr	CIDR of an ICM node's subnet within an existing private network.

4.2.3 Port and Protocol Parameters

Typically, the defaults for these parameters are sufficient. For information about two use cases in which you may need to specify some of these parameters, see Ports in the appendix "Using ICM with Custom and Third-Party Containers" and Ports in the appendix "Deploying on a Preexisting Cluster".

Parameter	Definition
ForwardPort	Port to be forwarded by a given load balancer (both 'from' and 'to'). Defaults: • AM: SuperServerPort parameter setting • WS: 80 • VM/CN: user provided; parameter must be included for a generic load balancer to be deployed
ForwardProtocol	Protocol to be forwarded by a given load balancer. Defaults: • AM: tcp • WS: http • VM/CN: user provided; parameter must be included for a generic load balancer to be deployed
HealthCheckPort	Port used to verify health of instances in the target pool. Defaults: • AM: SuperServerPort parameter setting • WS: 80 • VM/CN: user provided; parameter must be included for a generic load balancer to be deployed
HealthCheckProtocol	Protocol used to verify health of instances in the target pool. Defaults: • AM: tcp • WS: http • VM/CN: user provided; parameter must be included for a generic load balancer to be deployed
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; parameter must be included for a generic load balancer to be deployed
ISCAgentPort *	Port used by InterSystems IRIS ISC Agent. Default: 2188. If Containerless is false or absent and Overlay is set to weave (see General Parameters), this port is closed in the firewall.
SuperServerPort	Port used by InterSystems IRIS Superserver. Default: 1972.
WebServerPort	Port used by InterSystems IRIS Web Server/Management Portal. Default: 52773.
LicenseServerPort *	Port used by InterSystems IRIS License Server. Default: 4002.. If Containerless is false or absent and Overlay is set to weave (see General Parameters), this port is closed in the firewall.

* If ICM is in container mode (Containerless is false or absent) and Overlay is set to weave (see General Parameters), this port is closed in the node's firewall.

4.2.4 CPF Parameters

When using a CPF merge file specified by the UserCPF property to customize the CPF of one or more InterSystems IRIS instances during deployment, as described in Deploying with Customized InterSystems IRIS Configuration Parameters, you cannot include certain CPF settings, because ICM needs to read their values before it adds them to the CPF at a later stage. You should therefore customize these settings by specifying the following parameters (described in General Parameters and Port and Protocol Parameters) in your configuration files:

Parameter	CPF Setting
WIJMountPoint	[config]/wijdir
Journal1MountPoint	[Journal]/CurrentDirectory
Journal2MountPoint	[Journal]/AlternateDirectory
SuperServerPort	[Startup]/DefaultPort
WebServerPort	[Startup]/WebServerPort

Note: The value of the ICM LicenseServerPort field is taken from the **[LicenseServers]** block of the CPF, bound to the name of the configured license server (see InterSystems IRIS Licensing for ICM).

4.2.5 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 Serverices (AWS) Parameters
- Google Cloud Platform (GCP) Parameters
- Microsoft Azure (Azure) Parameters
- Tencent Cloud (Tencent) Parameters
- VMware vSphere (vSphere) Parameters

Note: For information about parameters used only for PreExisting deployments, see Definitions File for PreExisting in the appendix "Deploying on a Preexisting Cluster".

Some of the parameters listed are used with more than one provider; for example, the InstanceType, ElasticIP, and VPCId parameters can be used in both AWS and Tencent deployments. Some provider-specific parameters have different names but the same purpose, for example AMI and InstanceType for AWS, Image and MachineType for GCP, and ImageId and InstanceType for Tencent, whereas there are four Azure parameters corresponding to each of these.

Like the General Parameters table, the tables in this section indicate whether each parameter is required in every deployment or optional, and whether it must be included (when used) in either defaults.json or definitions.json, is recommended for one file or the other, or can be used in either. For examples of each type, see General Parameters.

4.2.5.1 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 (see the Region parameter in General Parameters). Another choice is which virtual machine images to use for the host nodes in your cluster (parameters vary by provider). 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.

Container images from InterSystems comply with the Open Container Initiative (OCI) specification, and are built using the Docker Enterprise Edition engine, which fully supports the OCI standard and allows for the images to be certified and featured in the Docker Hub registry. InterSystems images are built and tested using the widely popular container Ubuntu operating system, and ICM therefore supports their deployment on any OCI-compliant runtime engine on Linux-based operating systems, both on premises and in public clouds.

4.2.5.2 Amazon Web Services (AWS) Parameters

Parameter	Definition	Use is ...	Config file
Credentials	Path to a file containing the public/private keypair for an AWS account. To download, after logging into the AWS management console, open Managing Access Keys for IAM Users in the AWS documentation and follow the procedure for managing access keys in the AWS console.	required	defaults
AMI	AMI (machine image) to use as platform and OS template for nodes to be provisioned; see Amazon Machine Images (AMI) in the AWS documentation. Example: ami-a540a5e1. To list public AMIs available, in the EC2 Console, select **AMIs** in the navigation pane and filter for **Public AMIs**.	required	
InstanceType	Instance type to use as compute resources template for nodes to be provisioned on AWS and Tencent; see Amazon EC2 Instance Types in the AWS documentation. Example: m4.large. (Some instance types may not be compatible with some AMIs.)	required	
ElasticIP	Enables the Elastic IP feature on AWS and Tencent to preserve IP address and domain name across host node restart (see Host Node Restart and Recovery). Default: false.	optional	defaults
VPCId	Existing Virtual Private Cloud (VPC) to be used in the deployment on AWS and Tencent, instead of allocating a new one; the specified VPC is not deallocated during unprovision. If not specified when PrivateSubnet (see Security-related Parameters) is true, a new VPC is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network. **Note:** Internal parameter net_subnet_cidr must be provided if the VPC is not created in the default address space *10.0.%d.0/24*; for example, for a VPC in the range 172.17.0.0/24, you would need to specify net_subnet_cidr as 172.17.%d.0/24.	optional	defaults

Parameter	Definition	Use is ...	Config file
SubnetIds	When deploying on an existing private subnet on AWS or Tencent, comma-separated list of subnet IDs, one for each element specified by the Zone parameter (see General Parameters).	optional	defaults
RouteTableId	When deploying on an existing private subnet, theroute table to use for access to the ICM host; if provided, ICM uses this instead of allocating its own (and does not deallocate during unprovision). No default.	optional	defaults
InternetGatewayID	When deploying on an existing private subnet, the Internet gateway to use for access to the ICM host; if provided, ICM uses this instead of allocating its own (and does not deallocate during unprovision). No default.	optional	defaults
OSVolumeType	Determines disk type of the OS volume for a node or nodes in the deployment, which in turn determines the maximum value for the OSVolumeSize parameter (see General Parameters), which sets the size of the OS volume. See Amazon EBS Volume Types in the AWS documentation. Tencent uses the same parameter name. Default: standard.	optional	
DataVolumeType WIJVolumeType Journal1VolumeType Journal2VolumeType	Determines disk type of the corresponding persistent storage volume for **`iris`** containers (see Storage Volumes Mounted by ICM), which in turn determines the maximum size of the volume. For example, DataVolumeType determines the maximum value for the DataVolumeSize parameter (see General Parameters), which detemines the size of the data volume. See Amazon EBS Volume Types in the AWS documentation. Tencent uses the same parameter name. Default: standard.	optional	
OSVolumeIOPS	Determines IOPS count for the OS volume for a node or nodes in the deployment; see I/O Characteristics and Monitoring in the AWS documentation. Default: 0.	optional	
DataVolumeIOPS WIJVolumeIOPS Journal1VolumeIOPS Journal2VolumeIOPS	Determines IOPS count for the corresponding persistent storage volume for **`iris`** containers (see Storage Volumes Mounted by ICM). For example, DataVolumeIOPS determines the IOPS count for the data volume. See I/O Characteristics and Monitoring in the AWS documentation. Must be nonzero when the corresponding volume type (see the immediately preceding) is io1. Default: 0.	optional	

4.2.5.3 Google Cloud Platform (GCP) Parameters

Parameter	Definition	Use is ...	Config file
Credentials	Path to a JSON file containing the service account key for a GCP account. To download, after logging into the GCP console and selecting a project, open Creating and managing service account keys in the GCP documentation and follow the procedure for creating service account keys in the GCP console.	required	defaults
Project	GCP project ID; see Creating and Managing Projects in the GCP documentation.	required	defaults
Image	Source machine image to use as platform and OS template for provisioned nodes; see Images in the GCP documentation. Example: ubuntu-os-cloud/ubuntu-1804-bionic-v20190911.	required	
MachineType	Machine type to use as compute resources template for nodes to be provisioned; see Machine types in the GCP documentation. Example: n1-standard-1.	required	
RegionMap	When deploying across multiple regions (see Deploying Across Multiple Regions on GCP), specifies which nodes are deployed in which regions. Default: 0,1,2,...,255.	optional	definitions
Network	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 when PrivateSubnet (see Security-related Parameters) is true, a new VPC is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network.	optional	defaults
Subnet	Existing private subnet to be used in the deployment, instead of allocating a new one; not deallocated during unprovision. For multiregion deployments (see Deploying Across Multiple Regions on GCP), value must be a comma-separated list, one for each region specified. If not specified when PrivateSubnet (see Security-related Parameters) is true, a new VPC is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network.	optional	defaults
OSVolumeType	Determines disk type for the OS volume for a node or nodes in the deployment; see Storage Options in the GCP documentation. Default: pd-standard.	optional	
DockerVolume-Type	Determines disk type for the block storage device used for the Docker thin pool on a node or nodes in the deployment; see Storage Options in the GCP documentation. Default: pd-standard.	optional	

Parameter	Definition	Use is ...	Config file
DataVolumeType	Determines disk type for the corresponding persistent storage volume for **iris** containers (see Storage Volumes Mounted by ICM). For example, DataVolumeType determines the disk type for the data volume. See Storage Options in the GCP documentation. Default: pd-standard.	optional	
WIJVolumeType			
Journal1VolumeType			
Journal2VolumeType			

4.2.5.4 Microsoft Azure (Azure) Parameters

Parameter	Definition	Use is ...	Config file
SubscriptionId	A unique alphanumeric string that identifies a Microsoft Azure subscription; to display, on the Azure portal select **Subscriptions** or type "subscriptions" into the search box, and use the **Subscription ID** displayed for SubscriptionId.	required	defaults
TenantId	A unique alphanumeric string that identifies the Azure Active Directory directory in which an application was created; to display, on the Azure portal select **Azure Active Directory** in the nav pane and then **Properties** on the nav pane for that page, and use the **Directory ID** displayed for TenantId.	required	defaults
ClientId	Credentials identifying and providing access to an Azure application; to creat them: • Follow the procedure in Quickstart: Register an application with the Microsoft identity platform to create a new application registration. • Use the **Application ID** displayed on the App Registration tab for ClientId. • Select **Settings > Keys** to generate a key and use the key value displayed for ClientSecret.	required	defaults
ClientSecret			
Location	Region in which to provision a node or nodes; see the Region parameter in General Parameters.	required	defaults
LocationMap	When deploying across multiple regions (see Deploying Across Multiple Regions on Azure), specifies which nodes are deployed in which regions. Default: 0,1,2,...,255.	optional	definitions
PublisherName	Entity providing a given Azure machine image to use as platform and OS template for provisioned nodes. Example: OpenLogic.	required	
Offer	Operating system of a given Azure machine image. Example: UbuntuServer.	required	
Sku	Major version of the operating system of a given Azure machine image. Example: 7.2.	required	

Parameter	Definition	Use is ...	Config file
Version	Build version of a given Azure machine image. Example: 7.2.20170105.	required	
CustomImage	Image to be used to create the OS disk, in place of the Azure machine image described by the PublisherName, Offer, Sku, and Version fields. Value is an Azure URI of the form: /subscriptions/*subscription*/resource-Groups/*resource_group*/providers /Microsoft.Com-pute/images/*image_name*	optional	
Size	Machine size to use as compute resources template for nodes to be provisioned; see Sizes for virtual machines in Azure in the Azure documentation. Example: Standard_DS1.	required	
ResourceGroup-Name	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 when Pri-vateSubnet (see Security-related Parameters) is true, a new resource group is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network.	optional	defaults
VirtualNetworkName	Existing private subnet to be used in the deployment, instead of allocating a new one; not deallocated during unprovision. For multiregion deployments (see Deploying Across Multiple Regions on Azure), value must be a comma-separated list, one for each region specified. If not specified when PrivateSubnet (see Security-related Parameters) is true, a new VPC is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network. **Note:** The net_subnet_cidr parameter (see Security-related Parameters) must be provided if the net-work is not created in the default address space 10.0.%d.0/24.	optional	defaults

Parameter	Definition	Use is ...	Config file
SubnetName	Name of an existing subnet to be used in the deployment, instead of allocating a new one; not deallocated during unprovision. For multiregion deployments (see Deploying Across Multiple Regions on Azure), value must be a comma-separated list, one for each region specified. If not specified when PrivateSubnet (see Security-related Parameters) is true, a new subnet is allocated for the deployment and deallocated during unprovision. **Note:** When provisioning on a private network, unique SubnetName and net_subnet_cidr parameters must be provided for each entry in the definitions file (but ResourceGroupName and VirtualNetwork-Name remain in the defaults file). This includes the bastion host definition when deploying a bastion host (see Deploy on a Private Network Through a Bastion Host).	optional	definitions
AccountTier	Storage account performance tier (see Azure storage account overview in the Azure documentation); either HDD (Standard) or SSD (Premium).	optional	
AccountReplication-Type	Storage account replication type: locally-redundant storage (*LRS*), geo-redundant storage (*GRS*), zone-redundant storage (*ZRS*), or read access geo-redundant storage (*RAGRS*).	optional	

4.2.5.5 Tencent Cloud (Tencent) Parameters

Parameter	Definition	Use is ...	Config file
SecretID SecretKey	Unique alphanumeric strings that identify and provide access to a Tencent Cloud account. To download, open Signing Methods in the Tencent Cloud documentation and follow the procedure in "Applying for Security Credentials".	required	defaults
ImageId	Machine image to use as platform and OS template for provisioned nodes; see Image Overview in the Tencent documentation. Example: img-pi0ii46r.	required (see below)	
OSName	If ImageId (above) is not provided, ICM searches for an image matching this field. Note that this field supports regexp. Default: ubuntu.	required (see above)	
InstanceFamily	Instance family from which to select instance type; if InstanceType (below) is not provided, ICM searches for an instance type matching InstanceFamily, CPUCoreCount, and MemorySize (below). Default: S3.	required (see below)	

Parameter	Definition	Use is ...	Config file
InstanceType	Instance type to use as compute resources template for nodes to be provisioned on AWS and Tencent; see Instance Types in the Tencent documentation. Example: S2.MEDIUM4.	required (see above)	
ElasticIP	Enables the Elastic IP feature on AWS and Tencent to preserve IP address and domain name across host node restart (see Host Node Restart and Recovery). Default: false.	optional	defaults
VPCId	Existing Virtual Private Cloud (VPC) to be used in the deployment on AWS and Tencent, instead of allocating a new one; the specified VPC is not deallocated during unprovision. If not specified when PrivateSubnet (see Security-related Parameters) is true, a new VPC is allocated for the deployment and deallocated during unprovision. For more information, see Deploying Within an Existing Private Network. **Note:** Internal parameter net_subnet_cidr must be provided if the VPC is not created in the default address space *10.0.%d.0/24*; for example, for a VPC in the range 172.17.0.0/24, you would need to specify net_subnet_cidr as 172.17.%d.0/24.	optional	defaults
SubnetIds	When deploying on an existing private subnet on AWS or Tencent, comma-separated list of subnet IDs, one for each element specified by the Zone parameter (see General Parameters).	optional	defaults
CPUCoreCount	CPU core to match when selecting instance type; if InstanceType (above) is not provided, ICM searches for an instance type matching InstanceFamily, CPUCoreCount, and MemorySize (above). Default: 2.	optional	
MemorySize	Memory size to match when selecting instance type; if InstanceType (above) is not provided, ICM searches for an instance type matching InstanceFamily, CPUCoreCount, and MemorySize (above). Default: 4 GB.	optional	
OSVolumeType	Determines disk type for the OS volume for a node or nodes in the deployment; see Data Types: DataDisk in the Tencent documentation. AWS uses the same parameter name. Default: CLOUD_BASIC.	optional	
DockerVolume-Type	Determines disk type for the block storage device used for the Docker thin pool on a node or nodes in the deployment; see Data Types: DataDisk in the Tencent documentation. AWS uses the same parameter name. Default: CLOUD_BASIC.	optional	

Parameter	Definition	Use is ...	Config file
DataVolumeType	Determines disk type for the corresponding persistent storage volume for **`iris`** containers (see Storage Volumes Mounted by ICM). For example, DataVolumeType determines the disk type for the data volume. AWS uses the same parameter names. See Data Types: DataDisk in the Tencent documentation. Default: CLOUD BASIC.	optional	
WIJVolumeType			
Journal1Volume-Type			
Journal2Volume-Type			

4.2.5.6 VMware vSphere (vSphere) Parameters

Parameter	Definition	Use is ...	Config file
Server	Name of the vCenter server. Example: tbdvcenter.iscinternal.com.	required	defaults
Datacenter	Name of the datacenter.	required	defaults
DatastoreCluster	Collection of datastores where virtual machine files will be stored; see Creating a Datastore Cluster in the VMware documentation. Example: DatastoreCluster1.	required	defaults
VSphereUser	Credentials for vSphere operations; see About vSphere Authentication in the VMware documentation.	required	defaults
VSpherePassword			
DNSServers	List of DNS servers for the virtual network. Example: 172.16.96.1,172.17.15.53	required	defaults
DNSSuffixes	List of name resolution suffixes for the virtual network adapter. Example: iscinternal.com	required	defaults
Domain	FQDN for a node or nodes to be provisioned. Example: iscinternal.com	required	defaults
NetworkInterface	Label to assign to a network interface. Example: VM Network	optional	defaults
ResourcePool	Name of a vSphere resource pool; see Managing Resource Pools in the VMware documentation. Example: ResourcePool1.	optional	defaults
Template	Virtual machine master copy (machine image) to use as platform and OS template for nodes to be provisioned. Example: ubuntu1804lts	required	
VCPU	Number of CPUs in a node or nodes to be provisioned. Example: 2.	optional	
Memory	Amount of memory (in MB) in a node or nodes to be provisioned. Example: 4096.	optional	

Parameter	Definition	Use is ...	Config file
GuestID	Guest ID for the operating system type. See Enum - VirtualMachineGuestOsIdentifier on the VMware support website. Default: other3xLinux64Guest.	optional	
WaitForGuestNetTimeout	Time (in minutes) to wait for an available IP address on a virtual machine. Default: 5.	optional	
ShutdownWaitTimeout	Time (in minutes) to wait for graceful guest shutdown when making necessary updates to a virtual machine. Default: 3.	optional	
MigrateWaitTimeout	Time (in minutes) to wait for virtual machine migration to complete. Default: 10.	optional	
CloneTimeout	Time (in minutes) to wait for virtual machine cloning to complete. Default: 30.	optional	
CustomizeTimeout	Time (in minutes) that Terraform waits for customization to complete. Default: 10.	optional	
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.	optional	
SDRSEnabled	If specified, determines whether Storage DRS (see Enable and Disable Storage DRS in the VMware documentation) is enabled for a virtual machine; otherwise, use current datastore cluster settings. Default: Current datastore cluster settings.	optional	
SDRSAutomationLevel	If specified, determines Storage DRS automation level for a virtual machine; otherwise, use current datastore cluster settings. Values are automated or manual. Default: Current datastore cluster settings.	optional	

Parameter	Definition	Use is ...	Config file
SDRSIntraVMAffinity	If provided, determines Intra-VM affinity setting for a virtual machine (see Override VMDK Affinity Rules in the VMware documentation); 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.	optional	
SCSIControllerCount	Number of SCSI controllers for a given host node; 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	optional	
DockerVolumeSCSIController	SCSI controller on which to place the Docker volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1	optional	
DataVolumeSCSIController WIJVolumeSCSIController Journal1VolumeSCSIController Journal2VolumeSCSIController	SCSI controller on which to place the corresponding volume in `iris` containers; for example, DataVolumeSCSIController determines the controller for data volume. Must be between 1 and 4 and may not exceed SCSIControllerCount. Default: 1	optional	

Note: The requirements for the VMware vSphere template specified by the Template property are similar to those described in Host Node Requirements in the appendix "Deploying on a Preexisting Cluster" (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.

4.2.6 Device Name Parameters

The parameters listed in the following specify the device files under /dev that represent the persistent volumes created by ICM for use by InterSystems IRIS. For information about these persistent volumes and a table of provider and OS-specific default values for these parameters, see Storage Volumes Mounted by ICM. For PreExisting deployments, see Storage Volumes in the "Deploying on a Preexisting Cluster" appendix.

Parameter	Persistent Volume For
DataDeviceName	Databases
WIJDeviceName	WIJ directory
Journal1DeviceName	Primary journal directory
Journal2DeviceName	Alternate journal directory

4.2.7 Alphabetical List of User Parameters

The following table lists all of the parameters discussed in the preceding tables in this section in alphabetical order, with links to the table(s) containing their definition.

Parameter	Table(s) for definition
AccountReplicationType	Azure
AccountTier	Azure
AlternativeServers	General
AMI	AWS
ApplicationPath	General
ClientId	Azure, Security
ClientSecret	Azure, Security
CloneTimeout	vSphere
Count	General
CPUCoreCount	Tencent
Credentials	AWS, GCP, Security
CustomizeTimeout	vSphere
Datacenter	vSphere
DataDeviceName	Device Name

Parameter	Table(s) for definition
DataMountPoint	General
DatastoreCluster	vSphere
DataVolumeIOPS	AWS
DataVolumeSCSICon-troller	vSphere
DataVolumeSize	General
DataVolumeType	AWS, GCP, Tencent
DiskPolicy	vSphere
DNSName	PreExisting
DNSServers	vSphere
DNSSuffixes	vSphere
DockerImage	General
DockerPassword	General
DockerRegistry	General
DockerStorageDriver	General
DockerURL	General
DockerUsername	General
DockerVersion	General
DockerVolumeIOPS	AWS
DockerVolumeSCSICon-troller	vSphere
DockerVolumeSize	General
DockerVolumeType	AWS, GCP, Tencent
Domain	vSphere
ElasticIP	AWS, Tencent
FileSystem	General
GuestID	vSphere

Parameter	Table(s) for definition
Image	GCP
ImageId	Tencent
InstanceFamily	Tencent
InstanceType	AWS, Tencent
InternetGatewayID	AWS
IPAdress	PreExisting
ISCPassword	General
Journal1DeviceName	Device Name
Journal1MountPoint	General, CPF
Journal1VolumeIOPS	AWS
Journal1VolumeSCSIController	vSphere
Journal1VolumeSize	General
Journal1VolumeType	AWS, GCP, Tencent
Journal2DeviceName	Device Name
Journal2MountPoint	General, CPF
Journal2VolumeIOPS	AWS
Journal2VolumeSCSIController	vSphere
Journal2VolumeSize	General
Journal2VolumeType	AWS, GCP, Tencent
Label	General
LicenseDir	General
LicenseKey	General
LicenseServerPort	Port, CPF
LoadBalancer	General
Location	Azure

Parameter	Table(s) for definition
LocationMap	Azure
MachineType	GCP
Memory	vSphere
MemorySize	Tencent
MigrateWaitTimeout	vSphere
Mirror	General
MirrorMap	General
Namespace	General
NetworkInterface	vSphere
OSName	Tencent
OSVolumeIOPS	AWS
OSVolumeSize	General
OSVolumeType	AWS, GCP, Tencent
Overlay	General
Project	GCP
Provider	General
ProxyImage	General
Region	General
RegionMap	GCP
ResourceGroupName	Azure
ResourcePool	vSphere
Role	General
RouteTableId	AWS
SCSIControllerCount	vSphere
SDRSAutomationLevel	vSphere
SDRSEnabled	vSphere

Parameter	Table(s) for definition
SDRSIntraVMAffinity	vSphere
SecretID	Tencent, Security
SecretKey	Tencent, Security
Server	vSphere
ShutdownWaitTimeout	vSphere
Size	Azure
SSHOnly	Security
SSHPassword	Security
SSHPrivateKey	Security
SSHPublicKey	Security
SSHUser	Security
SSLConfig	Security
StartCount	General
SubnetName	Azure
SubnetIds	AWS, Tencent
SubscriptionId	Security
SuperServerPort	Port, CPF
SystemMode	General
Tag	General
Template	vSphere
TenantId	Security
TLSKeyDir	Security
UserCPF	General
VCPU	vSphere
VirtualNetworkName	Azure
VPCId	AWS, Tencent

Parameter	Table(s) for definition
VspherePassword	vSphere, Security
VsphereUser	vSphere, Security
WaitForGuestNetTimeout	vSphere
WebServerPort	Port, CPF
WIJDeviceName	Device Name
WIJMountPoint	General, CPF
WIJVolumeIOPS	AWS
WIJVolumeSCSICon-troller	vSphere
WIJVolumeSize	General
WIJVolumeType	AWS, GCP, Tencent
Zone	General
ZoneMap	General

4.3 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.

Table 4–3: ICM Node Types

Node Type	Configuration Role(s)	InterSystems Image to Deploy
DATA	Sharded cluster data node	iris (InterSystems IRIS instance)
COMPUTE	Sharded cluster compute node	iris (InterSystems IRIS instance)
DM	Distributed cache cluster data server Stand-alone InterSystems IRIS instance [namespace-level architecture: shard master data server]	iris (InterSystems IRIS instance)
DS	[namespace-level architecture: shard data server]	iris (InterSystems IRIS instance)

Node Type	Configuration Role(s)	InterSystems Image to Deploy
QS	[namespace-level architecture: shard query server]	iris (InterSystems IRIS instance)
AM	Distributed cache cluster application server [namespace-level architecture: shard master application server]	iris (InterSystems IRIS instance)
AR	Mirror arbiter	arbiter (InterSystems IRIS mirror arbiter)
WS	Web server	webgateway (InterSystems Web Gateway)
SAM	System Alerting and Monitoring (SAM) node	sam (InterSystems System Alerting and Monitoring application)
LB	Load balancer	—
VM	Virtual machine	—
CN	Custom and third-party container node	—
BH	Bastion host	—

Important: The InterSystems images shown in the preceding table are required on the corresponding node types, and cannot be deployed on nodes to which they do not correspond. If the wrong InterSystems image is specified for a node by the DockerImage field or the **-image** option of the **icm run** command — for example, if the **`iris`** image is specfied for an AR (arbiter) node, or any InterSystems image for a CN node — deployment fails, with an appropriate message from ICM. For a detailed discussion of the deployment of InterSystems images, see The icm run Command in the "Using ICM" chapter.

Note: The above table includes sharded cluster roles for the namespace-level sharding architecture, as documented in previous versions of this guide. These roles (DM, DS, QS) remain available for use in ICM but cannot be combined with DATA or COMPUTE nodes in the same deployment.

4.3.1 Role DATA: Sharded Cluster Data Node

As described in Overview of InterSystems IRIS Sharding and other sections of the "Horizontally Scaling for Data Volume with Sharding" chapter of the *Scalability Guide*, a typical sharded cluster consists only of data nodes, across which the sharded data is partitioned, and therefore requires only a DATA node definition in the definitions.json file. If DATA nodes are defined, the deployment must be a sharded cluster, and the only other node type that can be defined with them is COMPUTE.

DATA nodes can be mirrored if provisioned in a number matching the MirrorMap setting in their definition, as described in Rules for Mirroring. The DATA nodes in a cluster must be either all mirrored or all nonmirrored.

The only distinction between data nodes in a sharded cluster is that the first node configured (known as node 1) stores all of the nonsharded data, metadata, and code for the cluster in addition to its share of the sharded data. The difference in storage requirements, however, is typically very small. Because all data, metadata, and code is visible on any node in the cluster, application connections can be load balanced across all of the nodes to take greatest advantage of parallel query processing and partitioned caching. A load balancer may be assigned to DATA nodes; see Role LB: Load Balancer.

4.3.2 Role COMPUTE: Sharded Cluster Compute Node

For advanced use cases in which extremely low query latencies are required, potentially at odds with a constant influx of data, compute nodes can be added to a sharded cluster to provide a transparent caching layer for servicing queries, separating the query and data ingestion workloads and improving the performance of both. (For more information see Deploy Compute Nodes for Workload Separation and Increased Query Throughput in the *Scalability Guide*.)

Adding compute nodes yields significant performance improvement only when there is at least one compute node per data node, so you should define at least as many COMPUTE nodes as DATA nodes; if the number of DATA nodes in the definitions file is greater than the number of COMPUTE nodes, ICM issues a warning. Configuring multiple compute nodes per data node can further improve the cluster's query throughput, and the recommended best practice when doing so is to configure the same number of compute nodes for each data node, so ICM distributes the defined COMPUTE nodes as evenly as possible across the DATA nodes.

Because COMPUTE nodes support query execution only and do not store any data, their instance type and other settings can be tailored to suit those needs, for example by emphasizing memory and CPU and keeping storage to the bare minimum. Because they do not store data, COMPUTE nodes cannot be mirrored.

A load balancer may be assigned to COMPUTE nodes; see Role LB: Load Balancer.

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

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

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

Note: In an InterSystems IRIS sharded cluster with DS nodes (shard data servers) under the older namespace-level architecture, a single DM node serves as the shard master data server, providing application access to the shard data servers (DS nodes) on which the sharded data is stored and hosting nonsharded tables. (If shard master application servers [AM nodes] are included in the cluster, they provide application access instead.) A shard master data server can be mirrored by deploying two nodes of role DM and specifying mirroring.

4.3.4 Role DS: Shard Data Server

Under the namespace-level architecture, 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.

4.3.5 Role QS: Shard Query Server

Under the namespace-level architecture, shard query servers 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. If shard query 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.

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

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

A load balancer may be assigned to AM nodes; see Role LB: Load Balancer.

Note: When included in a sharded cluster with DS nodes (shard data servers) under the namespace-level architecture, 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.

4.3.7 Role AR: Mirror Arbiter

When DATA nodes (sharded cluster DATA nodes), a DM node (distributed cache cluster data server, stand-alone InterSystems IRIS instance, or namespace-level shard master data server), or DS nodes (namespace-level 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*.

4.3.8 Role WS: Web Server

A deployment 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 as follows:

- For a node-level sharded cluster:
  - If DATA and COMPUTE nodes are provisioned, all of the DATA and COMPUTE nodes.
  - If DATA nodes only are provisioned, all of the DATA nodes.
- For a namespace-level sharded cluster, a distributed cache cluster, or a standalone DM node:
  - If AM nodes are provisioned, all of the AM nodes,
  - If no AM nodes are provisioned, the DM node.

For mirrored DATA and DM nodes, a mirror-aware connection is created, and application connection redirection following failover is automatic. Communication between the web server and the remote servers is configured to run in TLS mode.

A load balancer may be assigned to WS nodes; see Role LB: Load Balancer.

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:2020.3.0.221.0",
    "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*.

4.3.9 Role SAM: System Alerting and Monitoring Node

Defining a SAM node adds the System Alerting and Monitoring (SAM) cluster monitoring solution to a deployment. For information about adding SAM, see Monitoring in ICM; for complete information about SAM, see the *System Alerting and Monitoring Guide*.

4.3.10 Role LB: Load Balancer

ICM automatically provisions a predefined load balancer node when the provisioning platform is AWS, GCP, Azure, or Tencent, and the definition of nodes of type DATA, COMPUTE, AM, or WS in the definitions file sets LoadBalancer to true. For a generic load balancer for VM or CN nodes, additional parameters must be provided.

4.3.10.1 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 data node in a sharded cluster or a distributed cache cluster application server.

To add a load balancer to the definition of DATA, COMPUTE, 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                                Region   Zone
-------           ----------    --------                                ------   ----
ANDY-AM-TEST-0001 54.214.230.24 ec2-54-214-230-24.amazonaws.com         us-west1 c
ANDY-AM-TEST-0002 54.214.230.25 ec2-54-214-230-25.amazonaws.com         us-west1 c
ANDY-LB-TEST-0000 (virtual AM)  ANDY-AM-TEST-1546467861.amazonaws.com  us-west1 c
```

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

The LoadBalancer field can be added to more than one definition in a deployment; for example a distributed cache cluster can contain AM nodes receiving load-balanced connections from and a WS tier receiving load-balanced application connections. Currently, a single automatically provisioned load balancer cannot serve multiple node types (for example, both DATA and COMPUTE nodes), so each requires its own load balancer. This does not preclude the user from manually deploying a custom or third-party load balancer to serve the desired roles.

4.3.10.2 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 the Ports and Protocol Parameters table 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.

Load balancers on different cloud providers may behave differently; be sure to acquaint yourself with load balancer details on platforms you provision on.

Avoid provisioning a load balancer for mirrored DATA nodes on Tencent; load balancers provisioned on Tencent are not currently able to determine which side of a mirrored DATA node is primary, which could result in errors performing read/write operations through the load balancer.

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

4.3.11 Role VM: Virtual Machine Node

A cluster may contain any number of virtual machine nodes. A virtual machine node provides a means of allocating host nodes 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 VM nodes; see Role LB: Load Balancer.

4.3.12 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 add InterSystems API Manager (IAM) to any deployment by defining a CN node and including the IAM and IAMImage fields; for more information, see Deploying InterSystems API Manager (IAM) in the "ICM Reference" chapter. You can also deploy any custom and third-party containers you wish on a CN node; **`iris`** (InterSystems IRIS) containers 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 either the **-role** or **-machine** option to limit the command to CN nodes (see ICM Commands and Options).

A load balancer may be assigned to CN nodes; see Role LB: Load Balancer. CN nodes cannot be deployed in containerless mode.

4.3.13 Role BH: Bastion Host

You may want to deploy a configuration that offers no public network access. If you have an existing private network, you can launch ICM on a node on that network and deploy within it. If you do not have such a network, you can have ICM configure a private subnet and deploy your configuration on it. Since ICM is not running within that private subnet, however, it needs a means of access to provision, deploy, and manage the configuration. The BH node serves this purpose.

A bastion host is a host node that belongs to both the private subnet configured by ICM and the public network, and can broker communication between them. To use one, you define a single BH node in your definitions file and set PrivateSubnet to true in your defaults file. For more information, see Deploying on a Private Network.

4.4 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 preventing configurations that are not functionally optimal, for example a single AM node, a single WS node, five DATA nodes with just one COMPUTE node or vice-versa, and so on.

In both nonmirrored and mirrored configurations,

- COMPUTE nodes are assigned to DATA nodes (and QS nodes to DS nodes) in round-robin fashion.
- If both AM and WS nodes are included, AM nodes are bound to the DM and WS nodes to the AM nodes; if just AM nodes or just WS nodes are included, they are all bound to the DM.

  Note: In this release, WS nodes are not compatible with a node-level sharded cluster and should not be included in such a deployment.

This section contains the following subsections:

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

4.4.1 Rules for Mirroring

All data nodes in a sharded cluster must be mirrored, or all unmirrored. This requirement 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, and the mirror roles of the DATA, DS, or DM nodes (primary, backup, or DR async) are determined by the value of the MirrorMap field (see General Parameters) in the node definition, as follows:

- If MirrorMap is not included in the relevant node definition, the nodes are configured as mirror failover pairs using the default MirrorMap value, **primary,backup**:
  - If an even number of DATA or DS nodes is defined, they are all configured as failover pairs; for example, specifying six DATA nodes deploys three data node mirrors containing failover pairs and no DR asyncs. If an odd number of DATA or DS nodes is defined, provisioning fails.
  - If two DM nodes are defined, they are configured as a failover pair; if any other number is defined, provisioning fails.

- If MirrorMap is included in the node definition, the nodes are configured according to its value, as follows:
  - The number of DATA or DS nodes must be a multiple of the number of roles specified in the MirrorMap value or fewer. For example, suppose the MirrorMap value, is **primary,backup,async**, as shown:

    ```
    "Role": "DATA",
    "Count": "",
    "MirrorMap": "primary,backup,async"
    ```

    In this case, DATA or DS nodes would be configured as follows:

Value of Count	Result
3 or multiples of 3	One or more mirrors containing a failover pair and a DR async
2	A single mirror containing a failover pair
1, 4 or more but not multiples of 3	Provisioning fails

  - The number of DM nodes must be the same as the number of roles specified in the MirrorMap value or fewer; if a single DM node is specified, provisioning fails.

- If more than one AR (arbiter) node is specified, provisioning fails. (While a best practice, use of an arbiter is optional, so an AR node need not be included in a mirrored configuration.)

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 *High Availability Guide*.

There is no relationship between the order in which DATA, DS, or DM nodes are provisioned or configured and their roles in a mirror. Following provisioning, you can determine which member of each pair is the intended primary failover member and which the backup using the **icm inventory** command. To see the mirror member status of each node in a deployed configuration when mirroring is enabled, use the **icm ps** command.

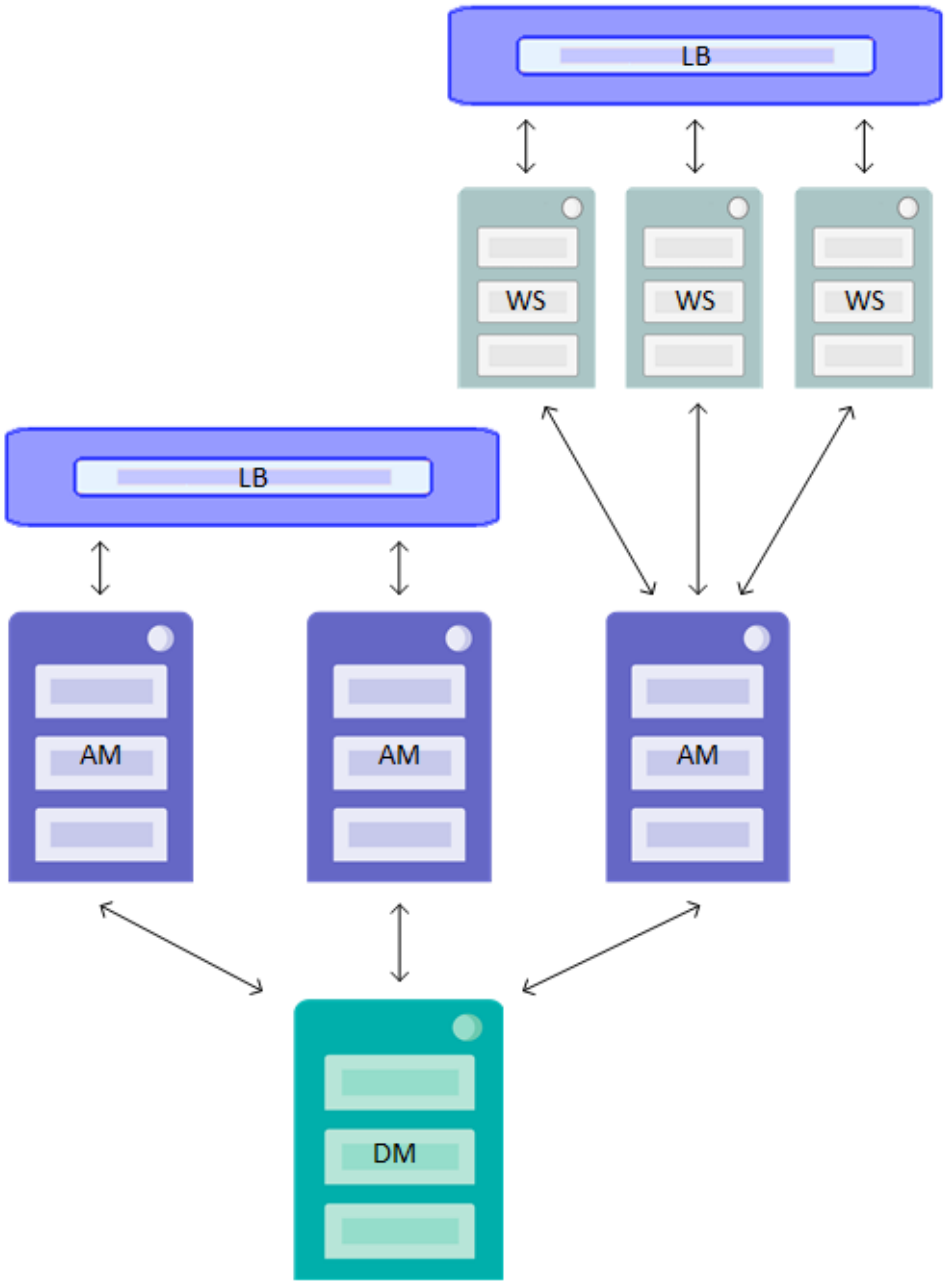
4.4.2 Nonmirrored Configuration Requirements

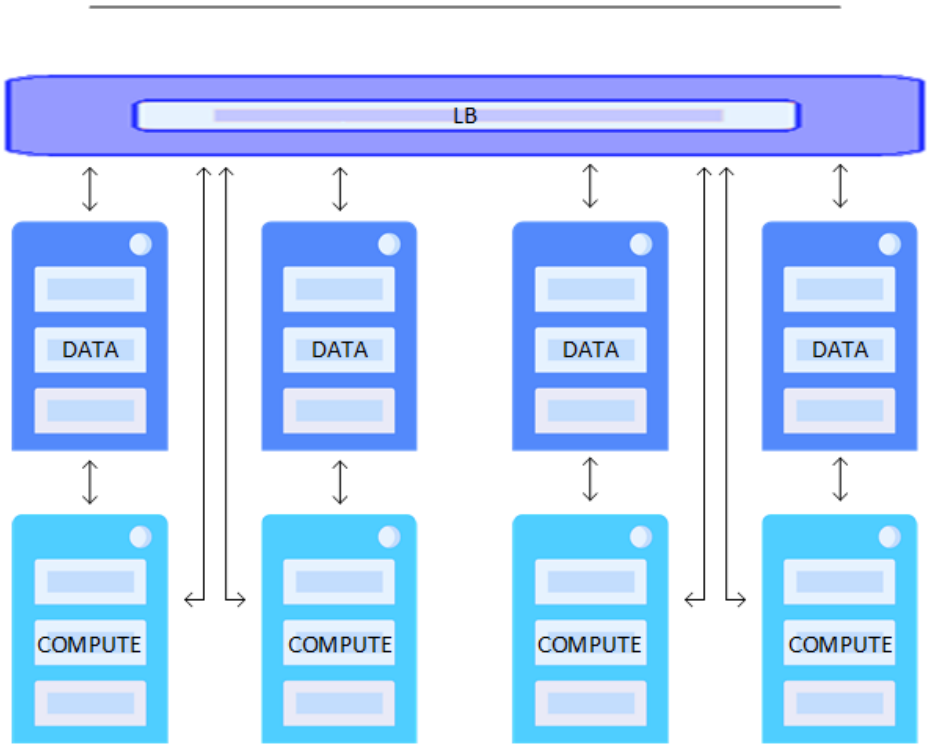
A nonmirrored cluster consists of the following:

- One or more DATA (data nodes in a sharded cluster).
- If DATA nodes are included, zero or more COMPUTE (compute nodes in a sharded cluster); best practices are at least as many COMPUTE nodes as DATA nodes and the same number of COMPUTE nodes for each DATA node.
- If no DATA nodes are included:
  - Exactly one DM (distributed cache cluster data server, standalone InterSystems IRIS instance, shard master data server in namespace-level sharded cluster).
  - Zero or more AM (distributed cache cluster application server, shard master application server in namespace-level sharded cluster).
  - Zero or more DS (shard data servers in namespace-level sharded cluster).
  - Zero or more QS (shard query servers in namespace-level sharded cluster).
- Zero or more WS (web servers).
- Zero or more LB (load balancers).
- Zero or more VM (virtual machine nodes).
- Zero or more CN (container nodes).
- Zero or one BH (bastion host).
- Zero AR (arbiter node is for mirrored configurations only).

The relationships between some of these nodes types are pictured in the following examples.

Figure 4–1: ICM Nonmirrored Topologies





4.4.3 Mirrored Configuration Requirements

A mirrored cluster consists of:

- If DATA nodes (data nodes in a node-level sharded cluster) are included:
  - A number of DATA matching the MirrorMap value, default or explicit, as described in Rules for Mirroring.
  - Zero or more COMPUTE (compute nodes in a node-level sharded cluster); best practices are at least one COMPUTE node per DATA node mirror, and the same number of COMPUTE nodes for each DATA node mirror.
- If no DATA nodes are included:
  - Two DM as a mirrored shard master data server in a namespace-level sharded cluster, data server in a distributed cache cluster, or standalone InterSystems IRIS instance, or more than two if DR asyncs are specified by the MirrorMap field, as described in Rules for Mirroring.
  - If a namespace-level sharded cluster:
    - A number of DS (shard data servers) matching the MirrorMap value, default or explicit, as described in Rules for Mirroring.
    - Zero or more QS (shard query servers), as described in the foregoing for COMPUTE nodes.
  - Zero or more AM as application servers in a distributed cache cluster or shard master application servers in a namespace-level sharded cluster.
- Zero or one AR (arbiter node is optional but recommended for mirrored configurations).
- Zero or more WS (web servers).
- Zero or more LB (load balancers).

- Zero or more VM (virtual machine nodes).
- Zero or more CN (container nodes).
- Zero or one BH (bastion host).

The following fields are required for mirroring:

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

  ```
  "Mirror": "true"
  ```

- To include DR asyncs in DATA, DS, or DM mirrors, you must include the MirrorMap field in your definitions file to specify that those beyond the first two are DR async members. The value of MirrorMap must always begin with **primary,backup**, for example:

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

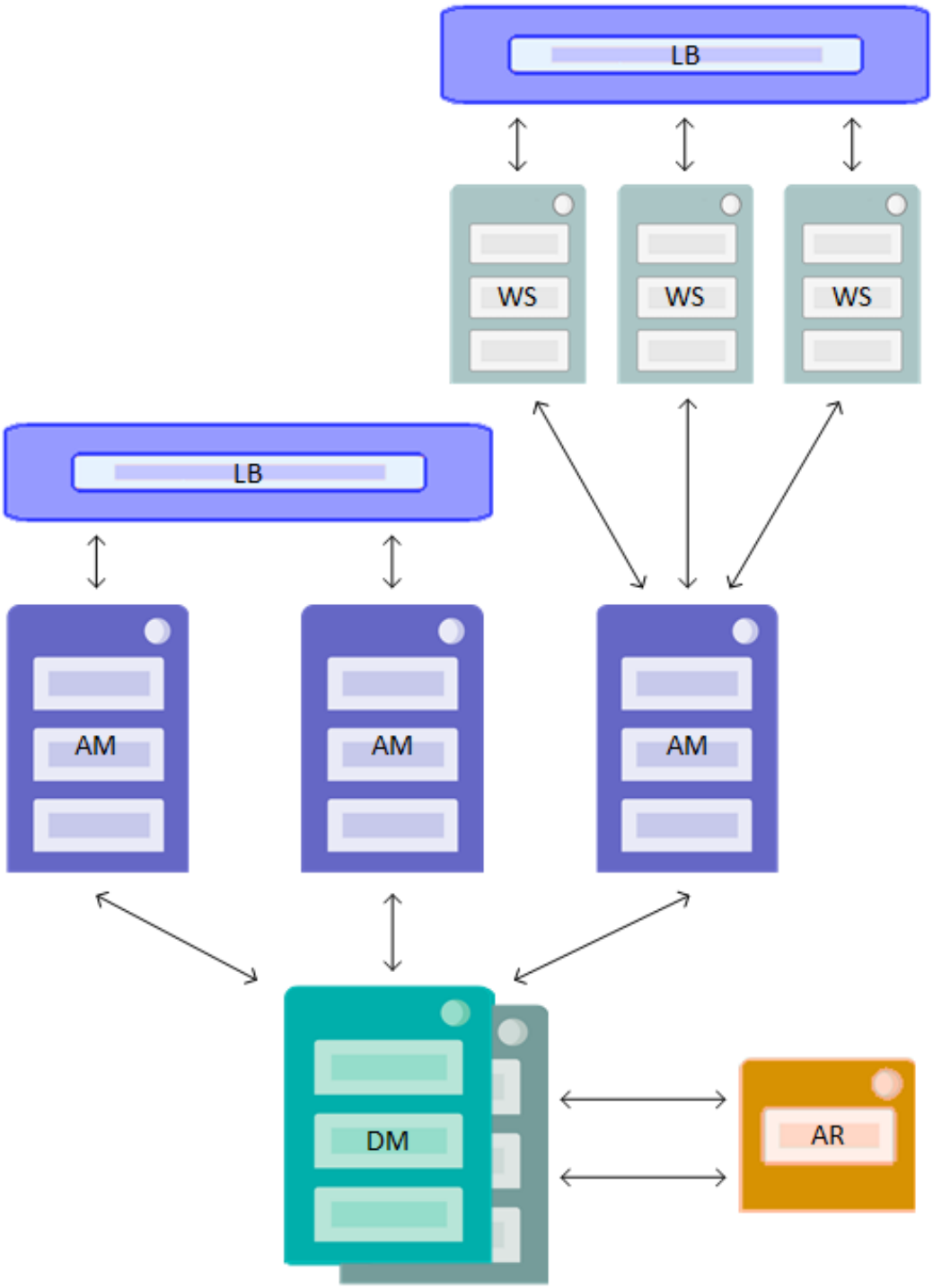
  For information on the relationship between the MirrorMap value and the number of DATA, DS, or DM nodes defined, see Rules for Mirroring. MirrorMap can be used in conjunction with the Zone and ZoneMap fields to deploy async instances across zones; see Deploying Across Multiple Zones.

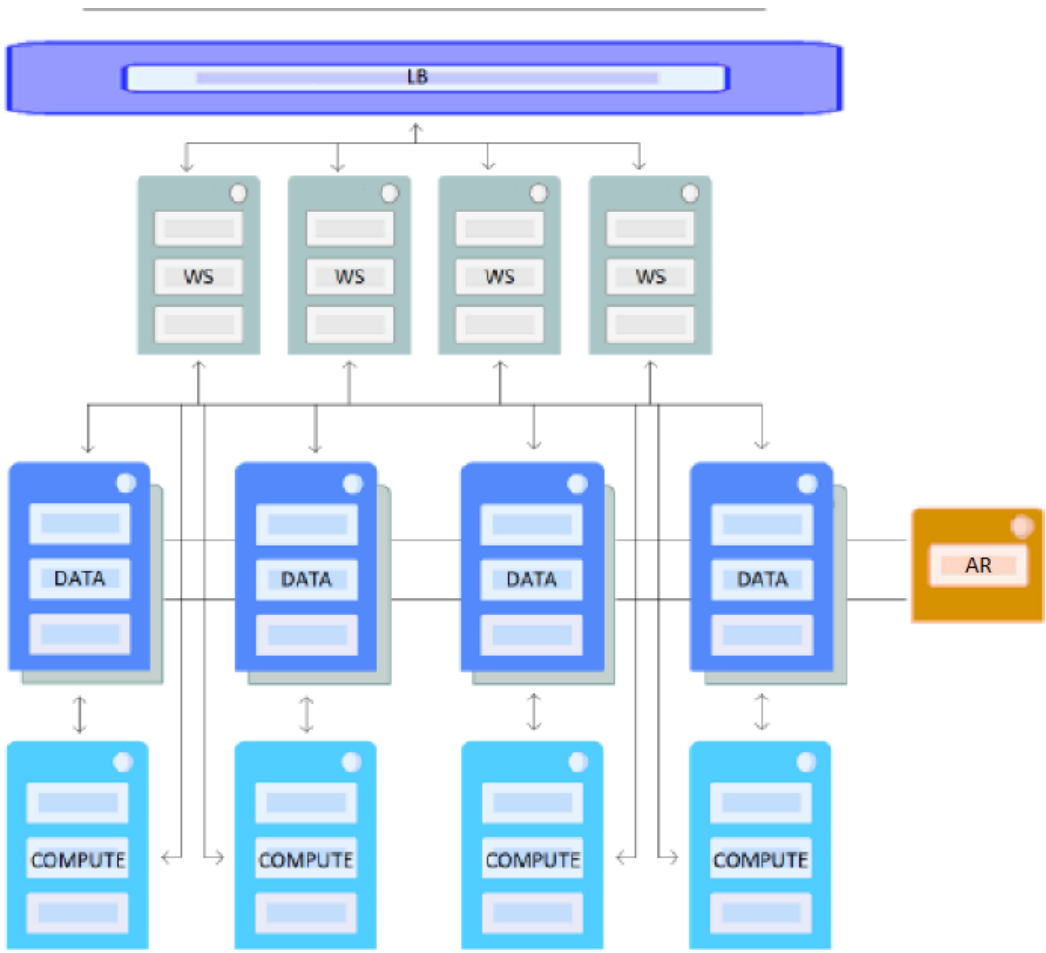
Automatic LB deployment (see Role LB: Load Balancer) is supported for providers AWS, GCP, Azure, and Tencent; when creating your own load balancer, the pool of IP addresses to include are those of DATA, COMPUTE, AM, or WS nodes, as called for by your configuration and application.

Note: A mirrored DM node that is deployed without AM or WS nodes or a load balancer (LB node) must have some appropriate mechanism for redirecting application connections following failover; see Redirecting Application Connections Following Failover or Disaster Recovery in the “Mirroring” chapter of the *High Availability Guide* for more information.

The relationships between some of these nodes types are pictured in the following examples.

Figure 4–2: ICM Mirrored Topologies





4.5 Storage Volumes Mounted by ICM

On each node on which it deploys an InterSystems IRIS container, ICM formats, partitions, and mounts four volumes for persistent data storage by InterSystems IRIS using the durable %SYS feature (see Durable %SYS for Persistent Instance Data in *Running InterSystems IRIS in Containers*). The volumes are mounted as separate device files under /dev/ on the host node, with the filenames determined by the fields DataDeviceName (for the data volume), WIJDeviceName (for the volume containing the WIJ directory), and Journal1DeviceName and Journal2DeviceName (for the primary and alternate journal directories). The sizes of these volumes can be specified using the DataVolumeSize, WIJVolumeSize, Journal1VolumeSize, and Journal2VolumeSize parameters (see General Parameters).

For all providers other than type PreExisting, ICM attempts to assign reasonable defaults for the device names, as shown in the following table. The values are highly platform and OS-specific, however, 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	Device Name of Persistent Volume for	AWS	GCP	Azure	Tencent	vSphere
DataDeviceName	Databases	xvdd	sdc	sdd	vdc	sdc
WIJDeviceName	WIJ directory	xvde	sdd	sde	vdd	sdd
Journal1Device-Name	Primary journal directory	xvdf	sde	sdf	vde	sde
Journal2Device-Name	Alternate journal directory	xvdg	sdf	sdg	vdf	sdf

This arrangement allows you to easily follow the recommended best practice of supporting performance and recoverability by using separate file systems for storage by InterSystems IRIS, as described in Separating File Systems for Containerized InterSystems IRIS in *Running InterSystems Products in Containers*.

Within the InterSystems IRIS container, ICM mounts the devices according to the fields shown in the following table:

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

4.6 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 field 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 DATA, COMPUTE, DM, AM, DS, and QS in the definitions.json file, for example:

```
"Role": "DM",
"LicenseKey": "ubuntu-sharding-iris.key”,
"InstanceType": "m4.xlarge",
```

- ICM configures a license server on DATA node 1 or the DM node, which serves the specified licenses to the InterSystems IRIS nodes (including itself) during deployment.

Important: All nodes on which an InterSystems IRIS container is deployed require a sharding-enabled InterSystems IRIS license, regardless of the particular configuration involved.

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. .

4.7 ICM Security

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

- Host Node Communication
- Docker
- Weave Net
- InterSystems IRIS
- Private Networks

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

4.7.1 Host Node Communication

A host node 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 host nodes and remotely execute commands on them, and SCP to copy files between the ICM container and a host 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 host 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.

4.7.2 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.

4.7.3 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.)

4.7.4 InterSystems IRIS

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

4.7.4.1 Security Level

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

4.7.4.2 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.

4.7.4.3 JDBC

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

4.7.4.4 Mirroring

ICM creates mirrors with 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 TLS enabled.

4.7.4.5 InterSystems Web Gateway

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

4.7.4.6 InterSystems ECP

ICM configures all InterSystems IRIS nodes to use TLS for ECP connections, which includes connections between distributed cache cluster nodes and sharded cluster nodes.

4.7.4.7 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*.

4.7.5 Private Networks

ICM can deploy on an existing private network (not accessible from the Internet) if you configure the access it requires. ICM can also create a private network on which to deploy and configure its own access through a bastion host. For more information on using private networks, see Deploying on a Private Network.

4.8 Deploying with Customized InterSystems IRIS Configurations

Every InterSystems IRIS instance, including the one running within an InterSystems IRIS container, is installed with a file in the installation directory named iris.cpf, which contains most of its configuration settings. The instance reads this *configuration parameter file*, or CPF, at startup to obtain the values for these settings. When a setting is modified, the CPF is automatically updated. The use and contents of the CPF are described in detail in the *Configuration Parameter File Reference*.

However, you may want to deploy multiple instances from the same image but with different configuration settings. You can do this using the **ISC_CPF_MERGE_FILE** environment variable, which lets you specify a separate file containing one or more settings to be merged into the CPF of an instance. The CPF merge feature can be used to deploy multiple instances with differing CPFs from the same source.

You can take advantage of this feature when deploying InterSystems IRIS with ICM by using the UserCPF property, which specifies the CPF merge file to be applied to **iris** containers or containerless installations. For example, the **[config]** section of the CPF included in InterSystems IRIS images from InterSystems contains the default generic memory heap configuration (see Configuring Generic Memory Heap in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*), which looks like this:

```
[config]
LibPath=
MaxServerConn=1
MaxServers=2
...
gmheap=37568
...
```

To double the size of the generic memory heap for all InterSystems IRIS instances in your deployment, you could create a file called merge.cpf in the ICM container with the following contents:

```
[config]
gmheap=75136
```

You would then specify this merge file in your defaults.json using the UserCPF field, as follows:

```
"UserCPF": "/Samples/mergefiles/merge.cpf"
```

This would cause the CPF of each InterSystems IRIS instance deployed to be updated with the new generic memory heap size before the instance is started.

You can also use this field in your definitions file to apply merge files only to specific node types. For example, to double the size of the generic memory heap only on the DM node in a distributed cache cluster, while at the same time changing the ECP **Time to wait for recovery** setting on the AM nodes from the default 1200 seconds to 1800, you would create another file called merge2.cpf with the following contents:

```
[ECP]
ClientReconnectDuration=1800
```

You would then use a definitions.json file like the following:

```
[
    {
        "Role": "DM",
        "Count": "1",
        "UserCPF": "/Samples/mergefiles/merge.cpf"
    },
    {
        "Role": "AM",
        "Count": "3",
        "StartCount": "2",
        "UserCPF": "/Samples/mergefiles/merge2.cpf",
        "LoadBalancer": "true"
    }
]
```

This would double the generic memory heap size on the DM node but not on the AM nodes, and change the ECP setting on the AM nodes but not on the DM node.

4.9 Deploying Across Multiple Zones

Cloud providers generally 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 mirrored sharded cluster in which each data node includes a failover pair and a DR async (see Mirrored Configuration Requirements), you can accomplish the cloud equivalent of putting physical DR asyncs in remote data centers by deploying the failover pair and the DR async in two different zones.

To specify multiple zones when deploying on AWS, GCP, Azure, and Tencent, 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"
```

For Tencent:

```
    "Provider": "Tencent",
    ...
    "Region": "na-siliconvalley",
    "Zone": "na-siliconvalley-1,na-siliconvalley-2"
```

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

For mirrored configuration, round-robin distribution may lead to undesirable results, however; for example, the preceding Zone specifications would place the primary and backup members of mirrored DATA, DM, or DS nodes in different zones, which might not be appropriate for your application due to higher latency between the members (see Network Latency

Considerations in 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

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

You could also use this approach with a mirrored sharded cluster in which each data node mirror contains a failover pair and a DR async, as follows:

defaults.json

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

definitions.json:

```
"Role": "DATA",
"Count": "12″,
"MirrorMap": "primary,backup,async",
"ZoneMap": "0,0,1",
...
"Role": "COMPUTE",
"Count": "8″,
"ZoneMap": "0",
...
"Role": "AR",
"ZoneMap": "2",
...
```

This would place the failover pair of each of the four data node mirrors and the eight compute nodes in us-west-1a, the DR async of each data node mirror in us-west-1b, and the arbiter in us-west-1c.

4.10 Deploying Across Multiple Regions or Providers

ICM can deploy across multiple cloud provider regions. For example, you may want to place DR async mirror members in a different region from their failover members. The procedures for multiregion deployment vary between providers, and are described in the following sections. The procedure described in the third section can also be used to deploy across multiple providers.

- Deploying Across Multiple Regions on GCP
- Deploying Across Multiple Regions on Azure
- Deploying Across Multiple Regions on AWS and Tencent

Important: Although the failover members of a mirror can be deployed in different regions or on different platforms, this is not recommended due to the problems in mirror operation caused by the typically high network latency between regions and platforms. For more information on latency considerations for mirrors, see Network Latency Considerations in the "Mirroring" chapter of the *High Availability Guide*.

Note: Deployment across regions and deployment on a private network, as described in Deploying on a Private Network, are not compatible in this release.

4.10.1 Deploying Across Multiple Regions on GCP

To deploy across multiple regions on GCP, specify the desired regions as a comma-separated list in the Region field in the defaults file, as shown:

```
{
    "Provider": "GCP",
    "Label": "Acme",
    "Tag": "multi",
    "Region": "us-east1,us-west1",
    "Zone": "us-east1-b,us-west1-a",
    ...
}
```

By default, nodes within each definition are assigned a region in round-robin fashion. For example, suppose you are deploying with the fields shown above in defaults.json and the following definitions.json:

```
[
  {
    "Role": "DATA",
    "Count": "2"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:2020.3.0.221.0"
  }
]
```

In this case, the output of the **icm inventory** command might look like this:

```
$ icm inventory
Machine               IP Address    DNS Name              Region    Zone
-------               ----------    --------              ------    ----
Acme-AR-multi-0001    35.179.173.90 acmear1.google.com    us-east1     b
Acme-DATA-multi-0001- 35.237.131.39 acmedata1.google.com  us-east1     b
Acme-DATA-multi-0002+ 35.233.223.64 acmedata2.google.com  us-west1     a
```

For control over the regions that nodes are deployed in, you can use the RegionMap field to map the defined nodes to the specified regions. When RegionMap is included in a node definition, ZoneMap (described in the preceding section, Deploying Across Multiple Zones) must also be included to map the node or nodes to the desired zone or zones. For example, suppose you are deploying a mirror containing a failover pair and a DR async with an arbiter, and you want the failover pair in one region but in different zones, and the async and arbiter in a different region and also in different zones. The files you might use and the output you might see from the **icm inventory** and **icm ps** commands are shown in the following:

defaults.json

```
{
    "Provider": "GCP",
    "Label": "Acme",
    "Tag": "multi",
    "Region": "us-east1,us-west1",
    "Zone": "us-east1-a,us-east1-b,us-west1-a,us-west1-b",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "RegionMap": "1,1,2",
    "ZoneMap": "1,2,3",
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:2020.3.0.221.0",
    "RegionMap": "2",
    "ZoneMap": "4"
  }
]
```

icm inventory

```
Machine               IP Address    DNS Name              Region    Zone
-------               ----------    --------              ------    ----
Acme-AR-multi-0001    35.179.173.90 acmear1.google.com    us-west1     b
Acme-DATA-multi-0001+ 35.237.131.39 acmedata1.google.com  us-east1     b
Acme-DATA-multi-0002- 35.233.223.64 acmedata2.google.com  us-east1     a
Acme-DATA-multi-0003  35.166.127.82 acmedata3.google.com  us-west1     a
```

icm ps

```
Machine              IP Address    Container Status Health  Mirror    Image
-------              ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001   35.179.173.90 arbiter   Up     healthy         intersystems/arbiter:2020.3.0.221.0
Acme-DATA-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:2020.3.0.221.0
Acme-DATA-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:2020.3.0.221.0
Acme-DATA-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:2020.3.0.221.0
```

To use the Network field (see Google Cloud Platform (GCP) Parameters) to specify an existing network to use in a multiregion deployment, you must also use the GCP Subnet field to specify a unique subnet for each region specified by the Region field. For example, for a deployment on regions us-west1 and us-east1, as illustrated here, you might include the following in your defaults file:

```
"Network": "acme-network",
"Subnet": "acme-subnet-data-east,acme-subnet-data-west"
```

Note: A GCP multiregion deployment cannot include load balancers (LB nodes) because load balancers are restricted to a single region on GCP.

4.10.2 Deploying Across Multiple Regions on Azure

To deploy across multiple regions on Azure, specify the desired regions as a comma-separated list in the Location field in the defaults file, as shown:

```
{
    "Provider": "Azure",
    "Label": "Acme",
    "Tag": "multi",
    "Location": "East US,Central US",
    ...
}
```

By default, nodes within each definition are assigned a location in round-robin fashion. For example, suppose you are deploying with the fields shown above in defaults.json and the following definitions.json:

```
[
  {
    "Role": "DATA",
    "Count": "2"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:2020.3.0.221.0"
  }
]
```

In this case, the output of the **icm inventory** command might look like this:

```
$ icm inventory
Machine               IP Address    DNS Name             Region    Zone
-------               ----------    --------             ------    ----
Acme-AR-multi-0001    35.179.173.90 acmear1.azure.com    East US    1
Acme-DATA-multi-0001- 35.237.131.39 acmedata1.azure.com  East US    1
Acme-DATA-multi-0001+ 35.233.223.64 acmedata2.azure.com  Central US 1
```

For control over the regions that nodes are deployed in, you can use the LocationMap field to map the defined nodes to the specified regions. When LocationMap is included in a node definition, ZoneMap (described in the preceding section, Deploying Across Multiple Zones) must also be included to map the node or nodes to the desired zone or zones. For example, suppose you are deploying a mirror containing a failover pair and a DR async with an arbiter, and you want the failover pair in one region but in different zones, and the async and arbiter in a different region and also in different zones. The files you might use and the output you might see from the **icm inventory** and **icm ps** commands are shown in the following. (Note that Azure zones are identified by the same integers in every region, so ZoneMap identifies only the desired zone within whatever region is specified by LocationMap.)

defaults.json

```
{
    "Provider": "Azure",
    "Label": "Acme",
    "Tag": "multi",
    "Location": "East US,Central US",
    "Zone": "1,2",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "LocationMap": "1,1,2",
    "ZoneMap": "1,2,1"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:2020.3.0.221.0",
    "RegionMap": "2",
    "ZoneMap": "2"
  }
]
```

icm inventory

```
Machine               IP Address    DNS Name             Region     Zone
-------               ----------    --------             ------     ----
Acme-AR-multi-0001    35.179.173.90 acmear1.azure.com    Central US 2
Acme-DATA-multi-0001+ 35.237.131.39 acmedata1.azure.com  East US    1
Acme-DATA-multi-0002- 35.233.223.64 acmedata2.azure.com  East US    2
Acme-DATA-multi-0003  35.166.127.82 acmedata3.google.com Central US 1
```

icm ps

```
Machine              IP Address    Container Status Health  Mirror    Image
-------              ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001   35.179.173.90 arbiter   Up     healthy         intersystems/arbiter:2020.3.0.221.0
Acme-DATA-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:2020.3.0.221.0
Acme-DATA-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:2020.3.0.221.0
Acme-DATA-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:2020.3.0.221.0
```

If you want to use an existing virtual network in a multiregion Azure deployment, you must include the ResourceGroupName and VirtualNetwork fields (see Microsoft Azure (Azure) Parameters) in the defaults file to specify a network for each region specified in the Location field, for example:

```
{
    "Provider": "Azure",
    "Label": "Acme",
    "Tag": "multi",
    "Location": "East US,Central US",
    "Zone": "1,2",
    "ResourceGroupName": "acme-resource-group",
    "VirtualNetworkName": "acme-vnet-east,acme-vnet-central"
    ...
}
```

The specified networks must have nonoverlapping address spaces. In the accompanying definitions file, each definition must include the Subnetname field specifying a unique subnet for each region specified by the Location field. For example, for the mirrored deployment illustrated in this section, if the defaults file included the ResourceGroupName and VirtualNetwork fields, the definitions file might look like the following. Because the AR definition deploys in the Central US region only, just one subnet is required in that definition.

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "RegionMap": "1,1,2",
    "ZoneMap": "1,2,1",
    "SubnetName": "acme-subnet-data-east,acme-subnet-data-central"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:2020.3.0.221.0",
    "RegionMap": "2",
    "ZoneMap": "2",
    "SubnetName": "acme-subnet-arbiter-central"
  }
]
```

Note: An Azure multiregion deployment cannot include load balancers (LB nodes) because load balancers are restricted to a single region on Azure.

4.10.3 Deploying Across Multiple Regions on AWS and Tencent

To deploy across multiple regions on AWS and Tencent, ICM first provisions the needed infrastructure in the separate regions, then merges that infrastructure and deploys services on it as if it were preexisting infrastructure.

This procedure can also be used to deploy across multiple providers. In this discussion, “region” is used to indicate “region or provider”, with differences between multiprovider and AWS/Tencent multiregion noted as needed.

The procedure for creating merged multiregion deployments involves the following steps:

1. Provision the infrastructure in each region in separate ICM sessions.
2. Merge the multiregion infrastructure using the **icm merge** command.
3. Review the merged definitions.json file to reorder and update as needed.
4. Reprovision the merged infrastructure using the **icm provision** command.
5. Deploy services on the merged infrastructure as a Preexisting deployment using the **icm run** command.

6. When unprovisioning the infrastructure, issue the **icm unprovision** command separately in the original session directories.

4.10.3.1 Provision the Infrastructure

The separate sessions for provisioning infrastructure in each region (specified by the Region field) should be conducted in separate working directories within the same ICM container. For example, you could begin by copying the provided /Samples/AWS directory (see Define the Deployment in the "Using ICM" chapter) to /Samples/AWS/us-east-1 and /Samples/AWS/us-west–1. Specify the desired region, node definitions, and features to match the eventual multiregion deployment in the defaulta and definitions file for each. For example, if you want to deploy a mirror failover pair in one region and a DR async member of the mirror in another, include the appropriate region and zones and **"Mirror": "true"** in the defaults files, and define two DMs (for the failover pair) in one region in its definitions file, a third DM (for the async) in the other, and a single AR (arbiter) node in one or the other. Each defaults file in a multiregion deployment should have a unique Label and/or Tag to prevent resource conflicts; this is not necessary for multiprovider deployments. This example is shown in the following.

Note: If a given definition doesn't satisfy topology requirements for a single-region deployment, for example a single DM node defined when Mirror is set to true, disable topology validation by including **"SkipTopologyValidation": "true"** in the defaults file, as shown in the /Samples/AWS/us-west-1/defaults.json.

/Samples/AWS/us-east-1

defaults.json

```
{
    "Provider": "AWS",
    "Label": "Acme",
    "Tag": "east1",
    "Region": "us-east-1",
    "Zone": "us-east1-a,us-east1-b",
    "Mirror": "true",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DM",
    "Count": "2",
    "ZoneMap": "1,2"
  }
]
```

/Samples/AWS/us-west-1/

defaults.json

```
{
    "Provider": "AWS",
    "Label": "Acme",
    "Tag": "west1",
    "Region": "us-west-1",
    "Zone": "us-west1-a,us-west1-b",
    "Mirror": "true",
    "SkipTopologyValidation": "true"},
    ...
}
```

definitions.json

```
[
  {
    "Role": "DM",
    "Count": "1",
    "ZoneMap": "1"
  }
  {
    "Role": "AR",
    "Count": "1",
    "ZoneMap": "2"
  }
]
```

Use the **icm provision** command in each working directory to provision the infrastructure in each region. The output of the **icm inventory** command, executed in each directory, shows you the infrastructure you are working with, for example:

/Samples/AWS/us-east-1

```
$ icm inventory
Machine             IP Address    DNS Name                        Region    Zone
-------             ----------    --------                        ------    ----
Acme-DM-east1-0001+ 54.214.230.24 ec2-54-214-230-24.amazonaws.com us-east-1 a
Acme-DM-east1-0002- 54.129.103.67 ec2-54-129-103-67.amazonaws.com us-east-1 b
```

/Samples/AWS/us-west-1

```
$ icm inventory
Machine             IP Address    DNS Name                        Region    Zone
-------             ----------    --------                        ------    ----
Acme-AR-west1-0001  54.181.212.79 ec2-54-181-212-79.amazonaws.com us-west-1 b
Acme-DM-west1-0002  54.253.103.21 ec2-54-253-103-21.amazonaws.com us-west-1 a
```

4.10.3.2 Merge the Provisioned Infractructure

The **icm merge** command scans the configuration files in the current working directory and those in the additional directory or directories specified to create merged configuration files that can be used for a Preexisting deployment in a specified new directory. For example, to merge the definitions and defaults files in /Samples/AWS/us-east–1 and /Samples/AWS/us-west–1 into a new set in /Samples/AWS/merge, you would issue the following commands:

```
$ cd /Samples/AWS/us-east-1
$ mkdir ../merge
$ icm merge -options ../us-west1 -localPath /Samples/AWS/merge
```

In the **icm merge** command, **--options** specifies a comma-separated list of the provisioning directories to be merged with the local one, and **--localPath** specifies the destination directory for the merged definitions.

4.10.3.3 Review the Merged Definitions File

When you examine the new configuration files, you will see that Provider has been changed to PreExisting in the merged defaults file. (The previous Provider field and others have been moved into the definitions file; they are displayed by the **icm inventory** command, but otherwise have no effect.) The Label and/or Tag can be modified if desired.

The definitions in the merged definitions file have been converted for use with provider PreExisting. As described in Definitions File for PreExisting in the appendix "Deploying on a Preexisting Cluster", the definitions.json file for a Preexisting deployment 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. Either the IPAddress or DNSName field must be included in each definition, as well as the SSHUser field. (The latter specifies a nonroot user with passwordless **sudo** access, as described in SSH in "Deploying on a Preexisting Cluster".) In the merged file, the definitions have been grouped by region, or by provider in multiprovider deployments; they should be reordered to reflect desired placement of mirror members, if necessary, and a suitable mirror map defined (see Mirrored Configuration Requirements and Deploying Across Multiple Zones). After review, the definitions file for our example would look like this:

```
[
    {
        "Role":"DM",
        "IPAddress":"54.214.230.24",
        "LicenseKey": "ubuntu-sharding-iris.key",
```

```
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"DM",
        "IPAddress":"54.129.103.67",
        "LicenseKey": "ubuntu-sharding-iris.key",
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"DM",
        "IPAddress":"54.253.103.21",
        "LicenseKey": "ubuntu-sharding-iris.key",
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"AR",
        "IPAddress":"54.181.212.79",
        "SSHUser": "icmuser",
        "StartCount": "4"
    }
]
```

4.10.3.4 Reprovision the Merged Infrastructure

Reprovision the merged infrastructure by issuing the **icm provision** command in the new directory (/Samples/AWS/merge in the example). The output of the **icm inventory** command shows the merged infrastructure in one list:

```
$ icm inventory
Machine             IP Address     DNS Name                        Region    Zone
-------             ----------     --------                        ------    ----
Acme-DM-east1-0001+ 54.214.230.24 ec2-54-214-230-24.amazonaws.com us-east-1 a
Acme-DM-east1-0002- 54.129.103.67 ec2-54-129-103-67.amazonaws.com us-east-1 b
Acme-AR-west1-0001  54.181.212.79 ec2-54-181-212-79.amazonaws.com us-west-1 b
Acme-DM-west1-0002  54.253.103.21 ec2-54-253-103-21.amazonaws.com us-west-1 a
```

4.10.3.5 Deploy Services on the Merged Infrastructure

Use the icm run command to deploy services on your merged infrastructure, as you would for any deployment, for example

```
$ icm run
...
-> Management Portal available at: http://112.97.196.104.google.com:52773/csp/sys/UtilHome.csp
$ icm ps
Machine            IP Address    Container Status Health  Mirror    Image
-------            ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001 35.179.173.90 arbiter   Up     healthy          intersystems/arbiter:2020.3.0.221.0
Acme-DM-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:2020.3.0.221.0
Acme-DM-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:2020.3.0.221.0
Acme-DM-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:2020.3.0.221.0
```

4.10.3.6 Unprovision the Merged Infrastructure

When the time comes to unprovision the multiregion deployment, return to the original working directories to issue the icm unprovision command, and then delete the merged working directory. In our example, you would do the following:

```
$ cd /Samples/AWS/us-east-1
$ icm unprovision -force -cleanUp
...
...completed destroy of Acme-east1
$ cd /Samples/AWS/us-west-1
$ icm unprovision -force -cleanUp
...
...completed destroy of Acme-west1
$ rm -rf /Samples/AWS/merge
```

4.11 Deploying on a Private Network

ICM configures the firewall on each host node to expose the only the ports and protocols required for its intended role. For example, the ISCAgent port is exposed only if mirroring is enabled and the role is one of AR, DATA, DM, or DS.

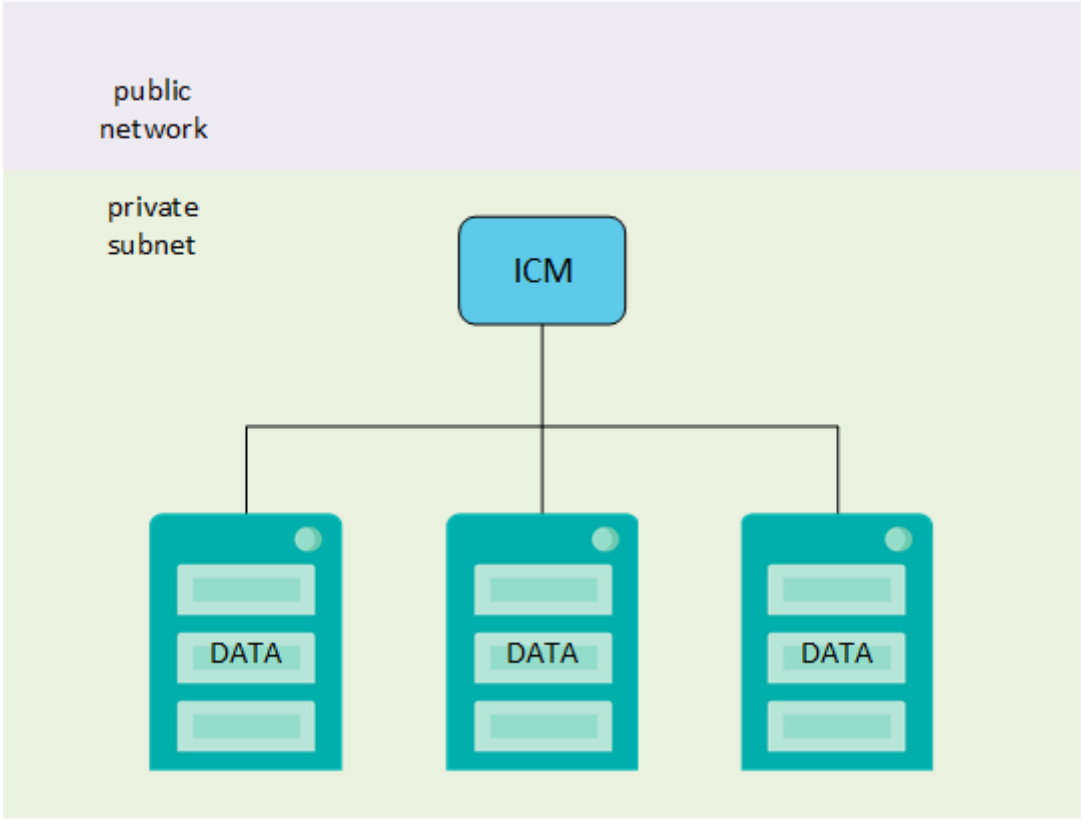
However, you may not want your configuration accessible from the public Internet at all. When this is the case, you can use ICM to deploy a configuration on a private network, so that it offers no direct public access. If ICM itself is deployed on that network, it is able to provision and deploy in the normal manner, but if it is not, you must provision a node outside the public network that gives ICM access to that network, called a *bastion host*. Given these factors, there are three approaches to using a private network:

- Install and run ICM within an existing private network, which you describe to ICM using several fields, some of which vary by provider.
- Have ICM provision a bastion host to give it access to the private network, and provision and deploy the configuration on either:
  - A private network created by ICM.
  - An existing private network, which you describe using the appropriate fields.

4.11.1 Deploy Within an Existing Private Network

If you deploy ICM on an existing private network and want to provision and deploy on that network, as shown in the following illustration, you need to add fields to the defaults and definitions files for the configuration you want to deploy.

Figure 4–3: ICM Deployed within Private Subnet



To deploy on an existing private network, follow these steps:

1. Obtain access to a node that resides within the private network. This may require use of a VPN or intermediate host.
2. Install Docker and ICM on the node as described in Launch ICM in the "Using ICM" chapter.
3. Add the following fields to the defaults.json file:

```
"PrivateSubnet": "true",
"net_vpc_cidr": "10.0.0.0/16",
"net_subnet_cidr": "10.0.2.0/24"
```

   The net_vpc_cidr and net_subnet_cidr fields (shown with sample values) specify the CIDRs of the private network and the node's subnet within that network, respectively.

4. Add the appropriate common and provider-specific fields to the defaults.json file, as follows:

Provider	Key	Description
all	PrivateSubnet	Must be set to true
	net_vpc_cidr	CIDR of the private network
	net_subnet_cidr	CIDR of the ICM node's subnet within the private network (see **Note)**
GCP	Network	Google VPC
	Subnet	Google subnetwork
Azure	ResourceGroupName	AzureRM resource group
	VirtualNetworkName	AzureRM virtual network
	SubnetName	AzureRM subnet (see **Note)**
AWS (see **Note**)	VPCId	AWS VPC ID
	SubnetIds	Comma-separated list of AWS subnet IDs, one for each element specified by the Zone field.
Tencent	VPCId	Tencent VPC ID
	SubnetIds	Comma-separated list of Tencent subnet IDs, one for each element specified by the Zone field.

Note: On Azure, ICM assigns a security group to the subnet specified by SubnetName, which could affect the behavior of unrelated machines on the subnet. For this reason, a dedicated subnet (as specified by a unique SubnetName and corresponding net_subnet_cidr) must be provided for every entry in the definitions file (but ResourceGroupName and VirtualNetworkName remain in the defaults file). This includes the BH definition when deploying a bastion host, as described in the following section.

To deploy IRIS within an existing private VPC on AWS, you must create a node within that VPC on which you can deploy and use ICM. If you want to reach this ICM host from outside the VPC, you can specify a route table and Internet gateway for ICM to use instead of creating its own. To do this, add the RouteTableId and InternetGatewayId fields to your defaults.json file, for example:

```
"RouteTableID": "rtb-00bef388a03747469",
"InternetGatewayId": "igw-027ad2d2b769344a3"
```

When provisioning on GCP, the net_subnet_cidr field is descriptive, not proscriptive; it should be an address space which includes the node's subnet, as well as any others within the network should have access to the deployed configuration.

5. Use **icm provision** and **icm run** to provision and deploy your configuration.

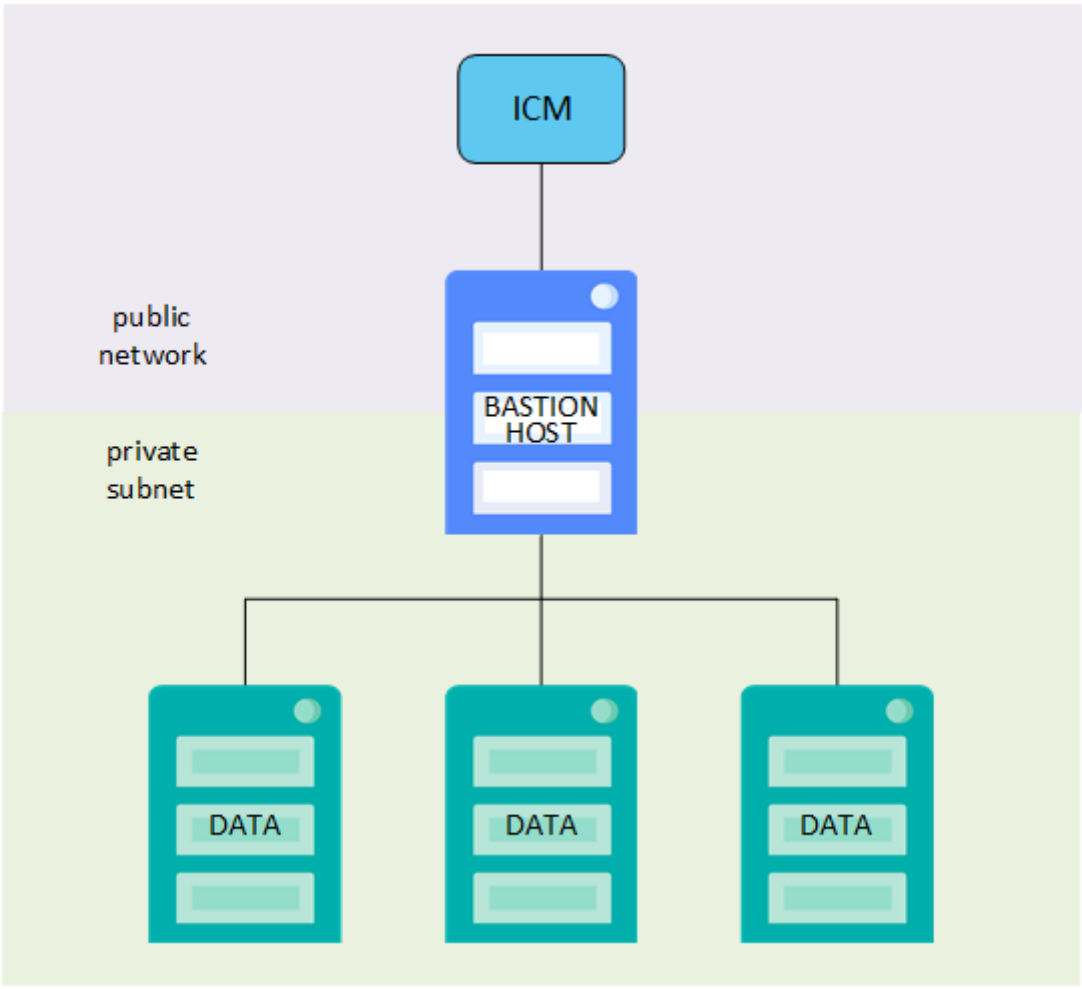
Bear the following in mind when deploying on a private network.

- Viewing web pages on any node within the private network, for example the Management Portal, requires a browser that also resides within the private network, or for which a proxy or VPN has been configured.
- Any DNS name shown in the output of ICM commands is just a copy of the local IP address.
- Private network deployment across regions or providers is currently not supported.

4.11.2 Deploy on a Private Network Through a Bastion Host

If you set the PrivateSubnet field to true in the defaults file but don't include the fields required to use an existing network, ICM creates a private network for you. You cannot complete the provisioning phase in this situation, however, because ICM is unable to configure or otherwise interact with the machines it just allocated. To enable its interaction with nodes on the private network it creates, ICM can optionally create a bastion host, a host node that belongs to both the private subnet and the public network and can broker communication between them.

Figure 4–4: ICM Deployed Outside a Private Network with a Bastion Host



To create a private network and a bastion host providing ICM with access to that network, add a definition for a single node of type BH to the definitions.json file, for example:

```
{
    "Role": "DATA",
    "Count": "3"
},
{
    "Role": "BH",
    "Count": "1",
    "StartCount: 4"
}
```

To deploy and use a bastion host with an existing private network, add a BH definition to the definitions file, as above, and include the fields necessary to specify the network in the defaults file (as describe in the previous section). ICM automatically sets the "PrivateSubnet" option to "true" when a BH node definition is included in definitions.json

The bastion host can be accessed using SSH, allowing users to tunnel SSH commands to the private network. Using this technique, ICM is able to allocate and configure compute instances within the private network from outside, allowing provisioning to succeed, for example:

```
$ icm inventory
Machine             IP Address     DNS Name                      Region   Zone
-------             ----------     --------                      ------   ----
Acme-BH-TEST-0004   35.237.125.218 218.125.237.35.bc.google.com us-east1 b
Acme-DATA-TEST-0001 10.0.0.2       10.0.0.2                      us-east1 b
Acme-DATA-TEST-0002 10.0.0.3       10.0.0.3                      us-east1 b
Acme-DATA-TEST-0003 10.0.0.4       10.0.0.4                      us-east1 b
```

Once the configuration is deployed, it is possible to run the *ssh* command against any node, for example:

```
# icm ssh -role DATA -interactive
ubuntu@ip-10.0.0.2:~$
```

If you examine the command being run, however, you can see that it is routed through the bastion host:

```
$ icm ssh -role DATA -interactive -verbose
ssh -A -t -i /Samples/ssh/insecure -p 3022 ubuntu@35.237.125.218
ubuntu@ip-10.0.0.2:~$
```

On the other hand, for other commands to succeed, ICM needs access to ports and protocols besides SSH. To do this, ICM configures tunnels between the bastion host and nodes within the cluster for Docker, JDBC, and HTTP. This allows commands such as icm run, icm exec, and icm sql to succeed.

Bear the following in mind when deploying a bastion host:

- The address of the configuration's Management Portal is that of the bastion host.
- For security reasons, no private keys are stored on the bastion host.
- Any DNS name shown in the output of ICM commands is just a copy of the local IP address.
- Provisioning of load balancers in a deployment that includes a bastion host is not supported.
- Use of a bastion host with multiregion deployments (see Deploying Across Multiple Regions or Providers) and in distributed management mode (see the appendix "Sharing ICM Deployments") are currently not supported.

  Note: When you create a custom VPC in the Google portal, you are required to create a default subnet. If you are provisioning with a bastion host and will use the subnet created by ICM, you should delete this default subnet before provisioning (or give it an address space that won't collide with the default address space 10.0.0.0/16).

4.12 Deploying InterSystems API Manager

The InterSystems API Manager (IAM) enables you to monitor and control traffic to and from your web-based APIs by routing it through a centralized gateway and forwarding API requests to appropriate target nodes. For complete information about IAM, see *Getting Started with API Manager* and *Administration Guide for API Manager*.

IAM is included in your ICM deployment when you define a CN node in your definitions file, include the IAM field with the value true, and specify the InterSystens **iam** image using the IAMImage field, for example:

```
[
    {
        "Role": "DATA",
        "Count": "1",
        "LicenseKey": "ubuntu-sharding-iris-with-iam.key"
    },
    {
        "Role": "CN",
        "Count": "1",
        "IAM": "true",
        "IAMImage": "intersystems/iam:2.0"
    }
]
```

The IAM container is deployed during the deployment phase (see The icm run Command). You can optionally also deploy a Postgres container by specifying a Postgres image using the PostgresImage field; its default value is shown in General Parameters.

Following successful deployment, a message like the example below is displayed:

```
$ icm run
...
-> IAM Portal available at: http://112.97.196.104.google.com:8080/overview#
```

IAM attaches to the InterSystems IRIS instance in the first (or only) **iris** container — for example, node 1 in a sharded cluster — to obtain an IAM-enabled IRIS license; if mirroring is enabled, this will be the primary of the first (or only) failover pair.

Note: IAM cannot be deployed in containerless mode.

4.13 Monitoring in ICM

To monitor the InterSystems IRIS instances in any ICM deployment, you can include the System Alerting and Monitoring cluster monitoring solution.

You can also deploy third-party monitoring packages as part of your ICM configuration.

4.13.1 System Alerting and Monitoring

System Alerting and Monitoring, or SAM, is a cluster monitoring solution for InterSystems IRIS® data platform. Whatever configuration and platform your InterSystems IRIS-based application runs on, you can monitor with SAM. For complete information about SAM, see the *System Alerting and Monitoring Guide*.

SAM is included in your ICM deployment when you include a SAM node in your definitions file; the DockerImage field is required and must specify the InterSystems **sam** image, as shown in the following:

```
[
    {
        "Role": "DM",
        "Count": "4",
        "LicenseKey": "ubuntu-sharding-iris.key"
    },
    {
        "Role": "SAM",
        "Count": "1",
        "DockerImage": "intersystems/sam:2.0"
    }
]
```

The SAM application comprises five containers. The SAM Manager container is deployed during the deployment phase (see The icm run Command).

The other four containers — Prometheus, Alertmanager, Grafana, and Nginx — are deployed during the provisioning phase (see The icm provision Command). The images from which these containers are deployed can be specified using the PrometheusImage, AlertmanagerImage, GrafanaImage, and NginxImage fields; their default values are shown in General Parameters.

Following successful deployment, a message like the example below is displayed:

```
$ icm run
...
-> SAM Portal available at: http://112.97.196.104.google.com:8080/api/sam/app/index.csp#
```

Note: SAM cannot be deployed in containerless mode.

4.13.2 Deploying Third-party Monitoring with ICM

You can deploy the third-party monitoring package of your choice (or any other third-party package) as part of your ICM configuration. The following example shows how to use the **icm ssh** command to add Weave Scope monitoring to all of the hosts in a deployment:

```
icm ssh -command "sudo curl -L git.io/scope -o /usr/local/bin/scope 2>&1"
icm ssh -command "sudo chmod +x /usr/local/bin/scope"
icm ssh -command "sudo /usr/local/bin/scope launch 2>&1"
```

Following these commands, you can access Weave Scope through port 4040 on any of the hosts displayed by the **icm inventory** command, that is, at http://*hostname*:4040.

Important: This simple example is provided for illustration only. Weave Scope does not require authentication and is therefore inherently insecure; if port 4040 is open in your firewall, anybody who knows the URL can access your containers. Do not deploy third-party packages outside of a private network unless you are certain they are fully secured.

4.14 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.

- Host 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

4.14.1 Host Node Restart and Recovery

When a cloud host 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 host node restart by default, whereas this feature is optional on AWS and Tencent (see Elastic IP Feature).

Reasons a host 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 host nodes include:

- Using the cloud provider user interface
- Using ICM:

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

4.14.1.1 Elastic IP Feature

The Elastic IP feature on AWS preserves IP addresses and domain names across host node restarts. ICM disables this feature by default, in part because it incurs additional charges on stopped machines (but not running ones). To enable this feature, set the ElasticIP field to true in your defaults.json file; be sure to review the feature for your provider (see Elastic IP Addresses in the AWS documentation or Elastic Public IP in the Tencent documentation).

4.14.1.2 Recovery and Restart Procedure

If the IP address and domain name of a host 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 Weave network deployed by ICM includes a decentralized discovery service, which means that if at least one host node has kept its original IP address, the other host nodes will be able to reach it and reestablish all of their connections with one another. However, if the IP address of every host node in the cluster has changed, an additional step is needed to connect all the nodes in the Weave network to a valid IP address.

The manual update 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
ANDY-DATA-TEST-0001	54.191.233.2	ec2-54-191-233-2.amazonaws.com
ANDY-DATA-TEST-0002	54.202.223.57	ec2-54-202-223-57.amazonaws.com
ANDY-DATA-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" : "SHARDING",
"Role" : "DATA",
"Tag" : "TEST",
"MachineName" : "ANDY-DATA-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                        Region   Zone
-------                ----------    --------                        ------   ----
ANDY-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
ANDY-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
ANDY-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
```

4. Use the icm ps command to verify that the host nodes are reachable:

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

5. If all of the IP addresses have changed, select one of the new 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 ANDY-DATA-TEST-0001...
Executing command 'weave connect 54.191.233.2' on host ANDY-DATA-TEST-0002...
Executing command 'weave connect 54.191.233.2' on host ANDY-DATA-TEST-0003...
...executed on ANDY-DATA-TEST-0001
...executed on ANDY-DATA-TEST-0002
...executed on ANDY-DATA-TEST-0003
```

4.14.2 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.)

4.14.3 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
```

4.14.4 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 host 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 host 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.

4.14.5 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.

4.14.6 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"
```

A

Containerless Deployment

If you want to use ICM to provision cloud infrastructure and install 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.

A.1 Containerless Deployment Platforms

Operating systems supported for containerless deployment include:

- Ubuntu 18.04 or later
- Red Hat Enterprise Linux 7.3 to 7.7
- SUSE Enterprise Linux 12 SP4, 15, and 15 SP1

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

- CentOS

A.2 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.3 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/2019.4.0/unix/IRIS-2019.4.0.792.0-lnxrhx64.tar.gz"
    },
    {
        "Role": "AM",
        "Count": "2",
        "StartCount": "2",
        "LoadBalancer": "true",
        "KitURL": "http://kits.acme.com/iris/2019.4.0/unix/IRIS-2019.4.0.792.0-lnxrhx64.tar.gz"
    }
]
```

In the defaults file:

```
{
    "Containerless": "true",
    "KitURL": "http://kits.acme.com/iris/2019.4.0/unix/IRIS-2019.4.0.792.0-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-2019.4.0.792.0-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-2019.4.0.792.0-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
```

Note: You can use the UserCPF field and a CPF merge file to install multiple instances with different configuration settings from the same kit; for more information, see Deploying with Customized InterSystems IRIS Configurations in the "ICM Reference" chapter.

A.4 Reinstalling InterSystems IRIS

To make the containerless deployment process as flexible and resilient as possible, the **icm install** command is fully reentrant — it can be issued multiple times for the same deployment. In this regard it is similar to the **icm run** command, as described in Redeploying Services in the "Using ICM" chapter.

When an **icm install** command is repeated, ICM stops and uninstalls the installed instances, then installs InterSystems IRIS from the specified kit again. You might want to repeat the command for one of the following reasons:

- Reinstalling the existing instances.

  To replace the installed InterSystems IRIS instances with new ones from the same kit, simply repeat the original **icm run** command that first deployed the containers. You might do this if you have made a change in the definitions files that requires reinstallation, for example you have updated the licenses in the directory specified by the LicenseDir field.

- Installing InterSystems IRIS on nodes you have added to the infrastructure, as described in Reprovisioning the Infrastructure.

  When you repeat an **icm install** command after adding nodes to the infrastructure, instances on the existing nodes are reinstalled as described in the preceding, while new instances are installed on the new nodes. This allows the existing nodes to be reconfigured for the new deployment topology, if necessary.

- Overcoming deployment errors.

  If the **icm install** command fails on one or more nodes due to factors outside ICM's control, such as network latency or interruptions, you can issue the command again; in most cases, deployment will succeed on repeated tries. If the error persists, however, it may require manual intervention — for example, if it is caused by an error in one of the configuration files — before you issue the command again.

A.5 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.6 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       2019.4.0.792.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2019.4.0.792.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2019.4.0.792.0  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**.

  Note: When the **icm ps** command is used prior to the **icm install** command in a containerless mode deployment, the **Status** field displays the value **not installed**.

- **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 the KitURL field.

  – 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       2019.4.0.792.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2019.4.0.792.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2019.4.0.792.0  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       2019.4.1.417.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2019.4.1.417.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2019.4.1.417.0  running  ok
```

After upgrade, you may want to update the value of KitURL in your defaults or definitions file to reflect the upgrade.

B

Sharing ICM Deployments

There are a number of situations in which different users on different systems might want to use ICM to manage or interact with the same deployment. For example, one user may be responsible for provisioning the infrastructure, while another in a different location is responsible for application deployment and upgrades.

However, ICM deployment is defined by its input files and results in the generation of several output files. Without access to these state files in the ICM container from which the deployment was made, it is difficult for anyone to manage or monitor the deployment (including the original deployer, should those files be lost).

To aid in this task, ICM can be run in distributed management mode, in which it stores the deployment's state files on a Consul cluster for access by other additional ICM containers. If distributed management mode is not used, state files can also be shared manually.

B.1 Sharing Deployments in Distributed Management Mode

ICM's distributed management mode uses the Consul service discovery tool from Hashicorp to give multiple users in any networked locations management access to a single ICM deployment. This is done through the use of multiple ICM containers, each of which includes a Consul client clustered with one or more Consul servers storing the needed state files.

- Distributed Management Mode Overview
- Configuring Distributed Management Mode
- Upgrading ICM Using Distributed Management Mode

B.1.1 Distributed Management Mode Overview

The initial ICM container, used to provision the infrastructure, is called the *primary ICM container* (or just the "primary ICM"). During the provisioning phase, the primary ICM does the following:

- Deploys 1, 3, or 5 Consul servers on CN nodes, which are clustered together with its Consul client.
- At the conclusion of the **icm provision** command,
  - Pushes the deployment's state files (see State Files for specifics) to the Consul cluster.
  - Outputs a **docker run** command for creating subsequent ICM containers for the deployment.

When a user executes the provided **docker run** command, a *secondary ICM container* (or "secondary ICM") is created, and an interactive container session is started in the provider-appropriate directory (for example, /Samples/GCP). The secondary ICM automatically pulls the deployment's state files from the Consul cluster at the start of every ICM command, so it always has the latest information. This creates a container that for all intents and purposes is a duplicate of the primary ICM container, with the one exception that it cannot provision or unprovision infrastructure. All other ICM commands are valid.

B.1.2 Configuring Distributed Management Mode

To create the primary ICM container and the Consul cluster, do the following:

1. Add the ConsulServers field to the defaults.json file to specify the number of Consul servers:

   ```
   "ConsulServers": "3"
   ```

   Possible values are 1, 3, and 5. A single Consul server represents a single point of failure and thus is not recommended. A five-server cluster is more reliable than a three-server cluster, but incurs greater cost.

2. Include a CN node definition in the definitions.json file specifying at least as many CN nodes as the value of the ConsulServers field, for example:

   ```
   {
        "Role": "CN",
        "Count": "3",
        "StartCount": "7",
        "InstanceType": "t2.small"
   }
   ```

3. Add the consul.sh script in the ICM container to the **docker run** command for the primary ICM, as follows:

   ```
   docker run --name primaryICM -it --cap-add SYS_TIME intersystems/icm:2020.3.0.221.0 consul.sh
   ```

When you issue the **icm provision** command on the primary ICM command line, a Consul server is deployed on each CN node as it is provisioned until the specified number of servers is reached. When the command concludes successfully, the primary ICM pushes the state files to the Consul cluster, and its output includes the secondary ICM creation command. When you subsequently issue any command in the primary ICM that might alter the instances.json file, such as **icm run** or **icm upgrade**, the primary ICM pushes the new file to the Consul cluster. When you use the **icm unprovision** command in the primary ICM to unprovision the deployment, its state files are removed from the Consul cluster.

The **icm run** command for the secondary ICM provided in output by **icm provision** includes an encryption key (16-bytes, Base64 encoded) allowing the new ICM container to join the Consul cluster, for example:

```
docker run -it --name ICM --cap-add SYS_TIME intersystems/icm:2020.3.0.221.0
  consul.sh qQ6MPKCH1YzTb0j9Yst33w==
```

You can use the secondary ICM creation command as many times as you wish, in any location that has network access to the deployment.

In both primary and secondary ICM containers, the **consul members** command can be used to display information about the Consul cluster, for example:

```
/Samples/GCP # consul members
Node                            Address              Status  Type    Build  Protocol DC  Segment
consul-Acme-CN-TEST-0002.weave.local  104.196.151.243:8301  failed  server  1.1.0  2        dc1 <all>
consul-Acme-CN-TEST-0003.weave.local  35.196.254.13:8301    alive   server  1.1.0  2        dc1 <all>
consul-Acme-CN-TEST-0004.weave.local  35.196.128.118:8301   alive   server  1.1.0  2        dc1 <all>
3be7366b4495                    172.17.0.4:8301      alive   client  1.1.0  2       dc1 <default>
e0e87449a610                    172.17.0.3:8301      alive   client  1.1.0  2       dc1 <default>
```

Consul containers are also included in the output of the **icm ps** command, as shown in the following:

```
Samples/GCP # icm ps
Pulling from consul cluster...
CurrentWorkingDirectory: /Samples/GCP
...pulled from consul cluster
Machine            IP Address       Container           Status   Health   Image
-------            ----------       ---------           ------   ------   -----
Acme-DM-TEST-0001  35.227.32.29     weave               Up                weaveworks/weave:2.3.0
Acme-DM-TEST-0001  35.227.32.29     weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-DM-TEST-0001  35.227.32.29     weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0004  35.196.128.118   consul              Up                consul:1.1.0
Acme-CN-TEST-0004  35.196.128.118   weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0004  35.196.128.118   weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0004  35.196.128.118   weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0002  104.196.151.243  consul              Up                consul:1.1.0
Acme-CN-TEST-0002  104.196.151.243  weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0002  104.196.151.243  weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0002  104.196.151.243  weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0003  35.196.254.13    consul              Up                consul:1.1.0
Acme-CN-TEST-0003  35.196.254.13    weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0003  35.196.254.13    weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0003  35.196.254.13    weavedb             Created           weaveworks/weavedb:2.3.0
```

Note: Because no concurrency control is applied to ICM commands, simultaneous conflicting commands issued in different ICM containers cannot all succeed; the results are based on timing and may include errors. For example, suppose two users in different containers simultaneously issue the command **icm rm -machine Acme-DM-TEST-0001**. One user will see this:

```
Removing container iris on Acme-DM-TEST-0001...
...removed container iris on Acme-DM-TEST-0001
```

while the other will see the following:

```
Removing container iris on Acme-DM-TEST-0001...
Error: No such container: iris
```

However, when no conflict exists, the same command can be run simultaneously without errors, for example **icm rm -machine Acme-DM-TEST-0001** and **icm rm -container customsensors -machine Acme-DM-TEST-0001**.

B.1.3 Upgrading ICM Using Distributed Management Mode

Because distributed management mode stores a deployment's state files on the Consul cluster, as described in Distributed Management Mode Overview, it provides an easy way to upgrade an ICM container without losing these files.

Beyond the benefits of having the latest version, upgrading ICM is necessary when you upgrade your InterSystems containers, because the major versions of the image from which you launch ICM and the InterSystems images you deploy must match. For example, you cannot deploy a 2019.4 version of InterSystems IRIS using a 2019.3 version of ICM. Therefore you must upgrade ICM before upgrading your InterSystems containers.

To upgrade an ICM container in distributed management mode, use these steps:

1. Use the secondary ICM **icm run** command provided at the end of provisioning by the primary ICM container, as described in Configuring Distributed Management Mode, to create a secondary ICM container from the ICM image you want to upgrade to. (Primary and secondary ICM containers created from different ICM images are compatible.)
2. In the primary ICM container, issue the command **consul.sh show-master-token** to get the value of the Consul token.
3. In the upgraded secondary ICM container, issue the command **consul.sh convert-to-thick** *Consul_token* to convert it to the primary ICM container.
4. Use **docker stop** and **docker rm** to stop and remove the old primary ICM container.

Because this is the recommended way to upgrade an ICM container that is managing a current deployment, you may want to create a primary ICM container every time you use ICM, whether you intend to use distributed management or not, so that this option is available.

Note: For information about upgrading a pre-2019.3 ICM container to release 2019.4, see the *Release Notes and Upgrade Checklist*.

B.2 Sharing Deployments Manually

This section explains how to share ICM deployments manually, describing which state files are required to share a deployment, methods for 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.2.1 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
- definitions.json

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
- state/ directory (can be overriden with **-stateDir** *path*)

The layout of the files under state/ 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.2.2 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.2.3 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.

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                        Region   Zone
-------                 ----------    --------                        ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
# 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                        Region   Zone
-------             ----------    --------                        ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
# 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 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                          Region   Zone
-------             ----------    --------                          ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
To destroy: icm unprovision [-cleanUp] [-force]

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

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                          Region   Zone
-------             ----------    --------                          ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b

# cat response.json
[
  {
    “Provider”: “AWS”,
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com",
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b”
  },
  {
    “Provider”: “AWS”,
    "MachineName ":"Acme-DATA-TEST-0002",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com,
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b””
  },
  {
    “Provider”: “AWS”,
    "MachineName":"Acme-DATA-TEST-0003",
    "Role":"DATA",
    "IPAddress":"54202.223.58",
    "DNSName":"54_202_223_58.amazonaws.com,
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b””
  }
]
```

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-DATA-TEST-0001  54.191.233.2   iris       Up      healthy isc/iris:2020.3.0.221.0
```

```
Acme-DATA-TEST-0002  54.202.223.57  iris       Up      healthy isc/iris:2020.3.0.221.0
Acme-DATA-TEST-0003  54.202.223.58  iris       Up      healthy isc/iris:2020.3.0.221.0
# cat response.json
[
  {
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:2020.3.0.221.0",
    "Status":"Up",
    "Health":"healthy"
  },
  {
    "MachineName ":"Acme-DATA-TEST-0002",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:2020.3.0.221.0",
    "Status":"Up"",
    "Health":"healthy"
  },
  {
    "MachineName ":"Acme-DATA-TEST-0003",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:2020.3.0.221.0",
    "Status":"Up",
    "Health":"healthy"
  },
]
```

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

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

# cat response.json
[
  {
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "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-DATA-TEST-0002",
    "Role":"DATA",
    "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-DATA-TEST-0003",
    "Role":"DATA",
    "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-DATA-TEST-0001...
ls: cannot access file.txt: No such file or directory
Error: See tmp/DATA-TEST/DATA-TEST-0001/ssh.err
Errors occurred during execution; aborting operation

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

# cat tmp/DATA-TEST/DATA-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.

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. This name is reserved and should be used only for containers derived from InterSystems IRIS 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 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 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 host 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.

E.1 Host Node Requirements for PreExisting

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

- SSH
- Ports
- Storage Volumes

Important: ICM cannot deploy containers on the host on which it is running. Because ICM has no way to determine the IP address of its host, it is the user's responsibility to avoid specifying the ICM host as a host node for Preexisting deployment.

E.1.1 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
  ```

  To prohibit password logins altogether, you can use set the SSHOnly parameter to true. Because this prevents ICM from logging in using a password, it requires that you stage your public SSH key (as specified by the SSHPublicKey field) on each node.

- 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 not secure.

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.1.2 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 Port and Protocol Parameters, the ports are configurable, for example:

```
"SuperServerPort": "51777"
```

If you change one or more of these fields from the defaults as illustrated, 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).
1972	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.

E.1.3 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 value **null** (literal string) for DataDeviceName, WIJDeviceName, Journal1DeviceName, or Journal2DeviceName causes ICM to simply create the mount point as a local directory on the host volume; this mode is not suitable for production use.

E.2 Definitions File for PreExisting

The primary 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 (as described in SSH, above).	icmuser