Oracle® TimesTen In-Memory Database

Kubernetes Operator User's Guide

Release 22.1
F35390-03
February 2023

Oracle TimesTen In-Memory Database Kubernetes Operator User's Guide, Release 22.1

F35390-03

Contents

Preface

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. The Oracle TimesTen Kubernetes Operator (TimesTen Operator) is a Kubernetes Operator that provides the ability to do the following:

- Create and deploy highly available replicated pairs of TimesTen Classic databases to a Kubernetes cluster with minimal effort. It also provides the ability to automate failure detection and recovery.
- Create and deploy TimesTen Scaleout grids and their associated databases to a Kubernetes cluster with minimal effort.

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and high throughput. The database resides entirely in memory at runtime and is persisted to the file system.

- Oracle TimesTen In-Memory Database in classic mode, or TimesTen Classic, refers to single-instance and replicated databases.
- Oracle TimesTen In-Memory Database in grid mode, or TimesTen Scaleout, refers to a multiple-instance distributed database. TimesTen Scaleout is a grid of interconnected hosts running instances that work together to provide fast access, fault tolerance, and high availability for in-memory data.
- TimesTen alone refers to both classic and grid modes (such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database).
- TimesTen Cache refers to a set of features that together enable the caching of performance-critical subsets of an Oracle database into cache tables within a TimesTen database for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard Structured Query Language (SQL) while data synchronization between the TimesTen database and the Oracle database is performed automatically.
- TimesTen Replication features, available with TimesTen Classic or TimesTen Cache, enable high availability.

TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET; Oracle interfaces PL/SQL, OCI, and Pro*C/C++; and the TimesTen TTClasses library for C++.

Audience

To work with this guide, you should understand how database systems work and have some knowledge of Kubernetes.

Related documents

TimesTen documentation is available on the TimesTen documentation website.

Oracle Database documentation is also available on the Oracle documentation website. This may be especially useful for Oracle Database features that TimesTen supports but does not attempt to fully document.

Conventions

The TimesTen Operator is supported in TimesTen Classic on the Linux x86-64 platform.

TimesTen Classic is supported on multiple platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX platforms. The term Linux is used separately.

TimesTen Scaleout is only supported on the Linux x86-64 platform. The information in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* applies only to this Linux platform.

See the *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory for specific platform versions supported by TimesTen.

Note:

In TimesTen documentation, the terms "data store" and "database" are equivalent. Both terms refer to the TimesTen database.

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value. For example: `LIBS = -Ltimesten_home/install/lib -ltten` Replace *`timesten_home`* with the path to the TimesTen instance home directory.
[ ]	Square brackets indicate that an item in a command line is optional.

Convention	Meaning
{ }	Curly braces indicate that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar separates alternative arguments.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
% or $	The percent sign or dollar sign indicates the UNIX shell prompt, depending on the shell that is used.
#	The number (or pound) sign indicates the prompt for the UNIX root user.

TimesTen documentation uses these variables to identify path, file and user names:

Convention	Meaning
`installation_dir`	The path that represents the directory where the current release of TimesTen is installed.
`timesten_home`	The path that represents the home directory of a TimesTen instance.
`release` or *`rr`*	The first two parts in a release number, with or without dots. The first two parts of a release number represent a major TimesTen release. For example, 221 or 22.1 represents TimesTen Release 22.1.
`DSN`	The data source name.

Note:

TimesTen release numbers are reflected in items such as TimesTen utility output, file names and directory names. The release numbers for these items are subject to change with every minor or patch release, and the documentation cannot always be up to date. The documentation seeks primarily to show the basic form of output, file names, directory names and other code that may include release numbers. You can confirm the current release number by looking at the Release Notes or executing the `ttVersion` utility.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

What's New

This section summarizes the new features of Oracle TimesTen In-Memory Database Release 22.1 that are documented in this guide. It provides links to more information.

New features in Release 22.1.1.9.0

- The TimesTen Operator monitors and manages TimesTen Scaleout objects that are deployed in your Kubernetes cluster. It also detects, repairs, and recovers from failures in your grid and associated database. See Managing TimesTen Scaleout.
- The Operator supports TimesTen Cache in TimesTen Scaleout. See Working with TimesTen Cache. For a complete example, see TimesTen Cache in TimesTen Scaleout Example.
- It is essential to specify memory requests and limits to Kubernetes. TimesTen recommends that CPU requests and limits be specified as well. See Specify CPU and Memory Requests and Limits. To support this functionality, there are new datum added to the `.spec.ttspec` fields of the TimesTenClassic and TimesTenScaleout object custom resource definitions. See TimesTenClassicSpecSpec and TimesTenScaleoutSpecSpec.
- TimesTen container images use the Oracle Linux base image. Oracle Java is installed into the TimesTen images using JDK script friendly URLs and Dockerfile techniques. For details about setting up your environment, see Setting Up the Environment . For specific information about TimesTen container images, see About TimesTen container images. For information about Dockerfile ARGS, see Dockerfile ARGs.

New features in Release 22.1.1.3.0

- The TimesTen Operator can deploy TimesTen Scaleout grids and their associated TimesTen databases in your Kubernetes cluster. See Deploying your TimesTen Grid.

  The TimesTen Operator supports the TimesTenScaleout object type. This object type provides the syntax you need to deploy a TimesTen Scaleout grid and database. See About the TimesTenScaleout object type.
- In previous releases, the TimesTen Operator required the creation of two container images, one for the Operator and one for TimesTen. In this release, one container image is used for both the Operator and TimesTen. You can create this container image or pull it from the Oracle Container Registry at `container-registry.oracle.com`. See Setting Up the Environment .
- When using the TimesTen Operator, the name of the Linux user that is created in the container image is `timesten` with a numeric UID of `3429`. The name of the Linux group that is created in the container image is `timesten` with a GID of `3429`. The `timesten` user is the user who runs TimesTen and the `timesten` group is the

TimesTen users group. You can override these defaults. This lets you tailor attributes of the image to meet your requirements. See Option 2b: Build with customizations and Dockerfile ARGs.

- The TimesTen Operator supports the TimesTen Prometheus Exporter. You can configure your TimesTenClassic and your TimesTenScaleout objects to use the Exporter. The Exporter can then collect metrics from the TimesTen databases that are running in your Kubernetes cluster, and expose these metrics to Prometheus. See Configuring with the TimesTen Prometheus Exporter.

  The TimesTen Operator provides the `prometheus` object type as part of the TimesTenClassic and TimesTenScaleout object type definitions. Use the `prometheus` object type to configure the Exporter to meet your Prometheus requirements. See TimesTenClassicSpecSpecPrometheus and TimesTenScaleoutSpecSpecPrometheus.

New features in Release 22.1.1.1.0

You can define a readiness probe to tell Kubernetes that a TimesTen (`tt`) container is ready. See Readiness probes.

1

Overview of the Oracle TimesTen Kubernetes Operator

This chapter provides an overview of containers and Kubernetes. It also gives an overview of the TimesTen Kubernetes Operator. It discusses the TimesTenClassic and TimesTenScaleout *Custom Resource Definitions* (CRDs) and the TimesTen Operator. The chapter details the role the TimesTen Operator plays in deploying, managing, and monitoring active standby pairs of TimesTen Classic databases as well as in deploying TimesTen Scaleout grids and their associated databases in your Kubernetes cluster.

Topics:

Overview of containers and Kubernetes

A container is a lightweight virtual machine, running the Linux operating system. A container usually runs one application that is started from an image. Files that are created and modified are usually not persistent. However, persistent storage is available. Containers are a key component of cloud computing environments.

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Kubernetes has the ability to manage the resources of multiple hosts (called *Nodes*) in a cluster, and to run containers as required across these nodes. It can automatically spawn containers to react to various failures. Kubernetes also manages the networking among the containers and to the outside world. Kubernetes is portable across many cloud and on-premises environments.

Key Kubernetes concepts include:

- *Pod*: One or more containers that share an IP address. For more information on Pods, see:

  `https://kubernetes.io/docs/concepts/workloads/pods/pod/`

- *Deployment*: A named collection of `n` identical Pods (where `n` is the number of Pods). Kubernetes ensures that `n` identical Pods are running. For more information on Deployments, see:

  `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`

- *PersistentVolume*: Storage that can be mounted to a Pod and is persistent beyond the lifetime of Pod. For more information on Persistent Volumes, see:

`https://kubernetes.io/docs/concepts/storage/persistent-volumes/`

- *StatefulSet*: Similar to a Deployment, but each Pod has an associated PersistentVolume. For more information on StatefulSets, see:

  `https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/`

- *Service*: A network endpoint for a Deployment or StatefulSet. It defines the set of addresses and ports that should be exposed to applications in the Kubernetes cluster. For more information on a Service, see:

  `https://kubernetes.io/docs/concepts/services-networking/service/`

Kubernetes provides the facilities for the provisioning of Pods and other Kubernetes resources that are required to deploy applications. Once deployed, the objects must be monitored and managed.

Kubernetes does some monitoring and managing of applications, but not all. It does handle problems at the Pod level automatically. For example, if a container fails, Kubernetes restarts it automatically. If an entire Node fails, Kubernetes starts replacement Pods on the other Nodes. However, Kubernetes has no knowledge about problems inside a container. This is not problematic for stateless applications, but for databases (which are stateful), Kubernetes needs help managing what is inside the containers.

This help comes in the form of:

- Custom Resource Definition
- Kubernetes Operator

Custom Resource Definition

A *Custom Resource Definition* (commonly known as a CRD) extends the Kubernetes' object model. It adds a new object type to the Kubernetes cluster, similar to the Pod, the StatefulSet, and the Service object types that it natively supports.

Kubernetes Operator

A *Kubernetes Operator* (also called *Operator*) is the brains behind a CRD. An Operator is an application that performs the functions of a human computer operator. It starts, stops, monitors, and manages other applications.

An Operator runs in one or more Pods, one active and the others idle. The active Operator performs the work. The remaining Operators are idle and remain idle until the active Operator fails. The active Operator manages all objects of a particular type and when combined with a CRD enables you to add custom facilities to Kubernetes.

About the TimesTen Kubernetes Operator

The TimesTen Kubernetes Operator provides the ability for you to deploy both active standby pairs of TimesTen Classic databases as well as TimesTen Scaleout grids and their associated databases in your Kubernetes cluster.

The TimesTen Kubernetes Operator consists of these interrelated components:

- Custom Resource Definitions (CRDs): There are two CRDs. The TimesTenClassic CRD defines an object of type TimesTenClassic to Kubernetes. This

TimesTenClassic object type provides the metadata for deploying active standby pairs of TimesTen Classic databases. The TimesTenScalout CRD defines an object of type TimesTenScaleout to Kubernetes. This TimesTenScaleout object type provides the metadata for deploying TimesTen Scaleout grids and their associated databases.

- TimesTen Operator: There is one TimesTen Operator. The Operator monitors the TimesTenClassic and TimesTenScaleout objects and properly handles both. It deploys, manages, and monitors active standby pairs of TimesTen Classic database. This same Operator deploys TimesTen Scaleout grids and their associated databases.
- TimesTen Agent: There is one TimesTen agent. This agent runs inside each container that runs TimesTen. The TimesTen Operator communicates with these agents both to determine the state of TimesTen insides of the container as well as to create, start, stop, and control TimesTen instances. The agent does not know how to manage TimesTen. It gives information to the Operator and the Operator provides the instructions for the agent. This agent knows how to work with TimesTen Classic and TimesTen Scaleout.

About TimesTenClassic and TimesTenScaleout objects

The TimesTen Operator distribution provides the file you need to deploy the TimesTenClassic and TimesTenScaleout CRDs in the Kubernetes cluster. Once deployed, Kubernetes understands the TimesTenClassic and TimesTenScaleout object types, just as it understands Pods, Secrets, and Services.

You can define objects of type TimesTenClassic or of type TimesTenScaleout or both. This lets you define the specific attributes for your TimesTen configuration and TimesTen database.

Objects in Kubernetes are named and typed. You can define a TimesTenClassic object named `sample` and another TimesTenClassic object named `sample2`. Similarly, you can define a TimesTenScaleout object named `sample` and another TimesTenScaleout object named `sample2`. You can have as many of these Kubernetes objects as you want, limited only by the available resources in your Kubernetes cluster.

Objects of different types have different meanings. For example, an object of type TimesTenClassic has a different meaning than an object of type ConfigMap. Therefore, you can define a `sample` TimesTenClassic object and a `sample` ConfigMap. The same is true for TimesTenScaleout objects.

Kubernetes supports *namespaces*. Namespaces split a cluster into multiple independent ones. Each namespace has a completely independent set of names. There can be an object of type `a` called `x` in `namespace1` and a different object of type `a` called `x` in `namespace2`. For example, you can define an object of type TimesTenClassic called `sample` in the `namespace1` namespace, and a different object of type TimesTenClassic called `sample` in the `namespace2` namespace. The same is true for TimesTenScaleout objects.

Note:

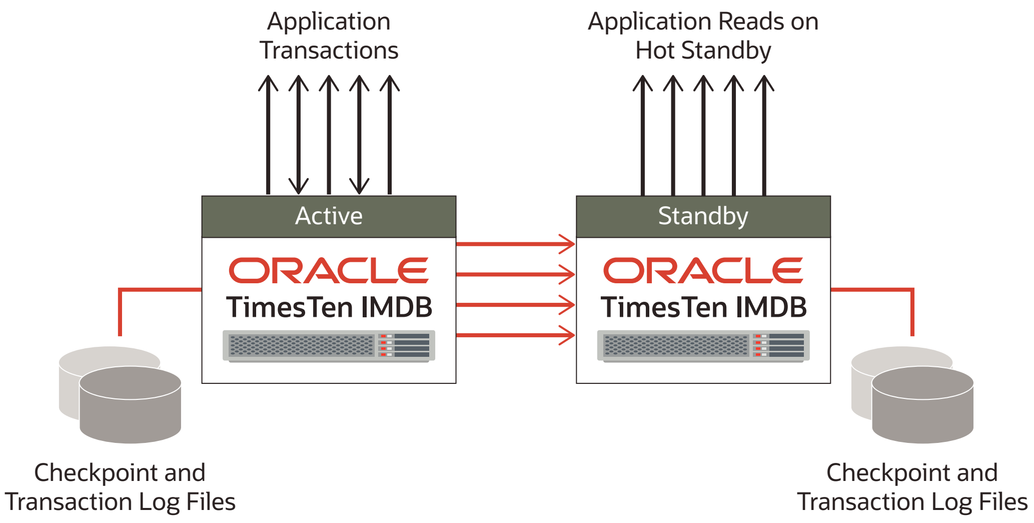
CRDs are cluster-scoped, not namespace-scoped.

Kubernetes object definitions are expressed in JSON or YAML. The examples in this book use YAML.

About provisioning active standby pairs

TimesTen Classic databases almost always run in active standby pairs. Figure 1-1 illustrates an active standby pair replication scheme. There are two databases. One database is the active, and the second database is the standby. Applications update the active database. The standby database is read-only and receives replicated updates from the active database. Only one of the two databases function as the active database at any one time. If the active database fails, the standby database is promoted to be the active. After the failed (active) database is recovered, it becomes the standby database. See "Types of replication schemes" in the *Oracle TimesTen In-Memory Database Replication Guide* for more information on the active standby pair replication scheme.

Figure 1-1 Active standby pair of TimesTen databases



An active standby pair replication scheme is a good fit for Kubernetes. Specifically, consider a pair of Pods, each with persistent storage, that are running an active standby pair of TimesTen databases. If the Pod containing the active database fails, Kubernetes automatically spawns another Pod to take its place, and attaches the appropriate storage to it.

But, since Kubernetes doesn't know anything about TimesTen, it will not automatically perform the necessary operations to cause the standby database on the surviving Pod to become the active database. This is where the TimesTen Operator comes in.

TimesTen provides a CRD that adds the *TimesTenClassic* object type to Kubernetes as well as an Operator for managing TimesTen databases. The Operator automates setup and initial configuration, manages databases and replication, and automates failover and recovery.

When you define a TimesTenClassic object, you can specify the configuration of your TimesTen deployment using Kubernetes facilities. When you create a TimesTenClassic object in a Kubernetes cluster, a pair of Pods are created, each running TimesTen. Each Pod contains a TimesTen instance. Each instance provides one TimesTen database. Database replication, through active standby pairs, is automatically configured. In short, you can deploy highly available replicated pairs of TimesTen databases and manage them by issuing a small number of commands.

A Kubernetes Operator manages objects of a particular type. TimesTen provides an Operator that manages Kubernetes objects of type TimesTenClassic. In so doing, TimesTen can be deployed, monitored, managed, and controlled in an automated manner with no required human intervention.

About deploying a TimesTenClassic object

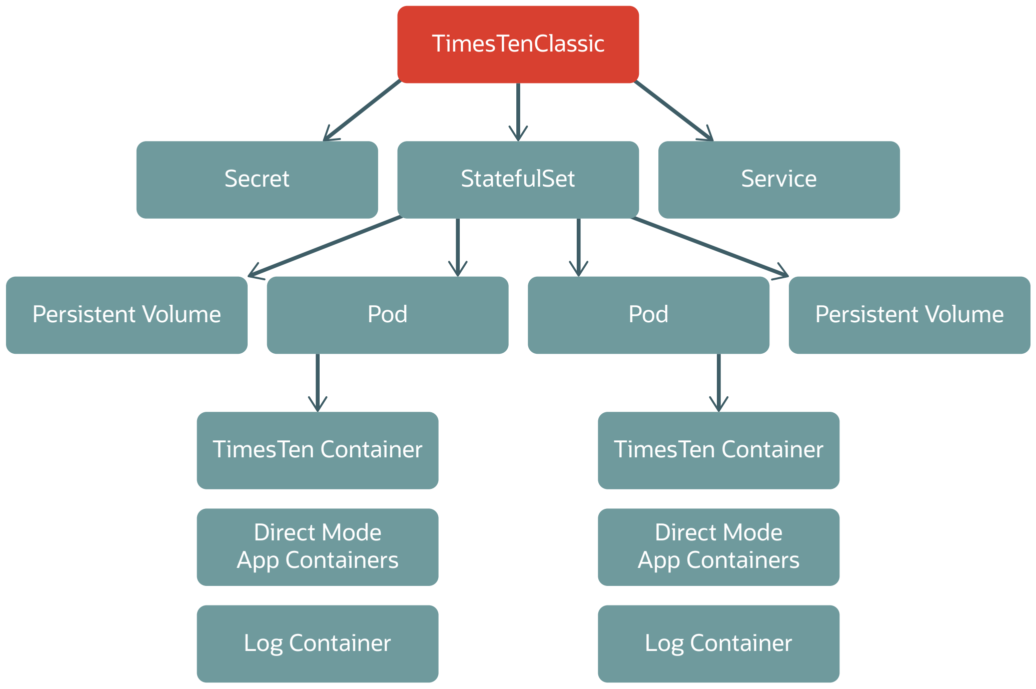
When you create a TimesTenClassic object in the Kubernetes cluster, the process to create and configure your active standby pair of databases begins. The Operator is invoked and creates several Kubernetes objects that are required to run TimesTen. After the objects are created and linked together, the TimesTen containers run a script to configure and start the TimesTen agent. The Operator communicates with the TimesTen agent that is running in each Pod in order to monitor and control TimesTen. The Operator configures one database as the active database, copies the active database to the standby, and then configures the active standby pair replication scheme. The process is described in detail in these sections:

- About objects created by the TimesTen Operator
- About the TimesTen containers and the TimesTen agent
- Simple deployment

About objects created by the TimesTen Operator

The Operator creates a number of Kubernetes objects that are required to run TimesTen, including a StatefulSet, a Service, and a Secret. These objects in turn create other objects. All of these objects are linked together by Kubernetes and are associated with the TimesTenClassic object you created. Figure 1-2 shows the objects that are created and how they are linked together.

Figure 1-2 Creating the TimesTenClassic object



The objects that are created are described in the following sections:

- StatefulSet
- Service
- Secret
- Pods
- Events

StatefulSet

The Operator creates a StatefulSet consisting of two Pods to run TimesTen. Each Pod has one or more PersistentVolumes (persistent storage), that are mounted in the TimesTen containers. This is where your TimesTen databases are stored. Applications running in the containers with PersistentVolumes mounted can create files that live beyond the lifetime of the container. (By default, all files that containers create and modify automatically vanish when the container exits. Containers are ephemeral.)

One attribute of a StatefulSet is the number of `replicas` that can be provisioned. Each TimesTenClassic object has an associated StatefulSet with two `replicas`. If one Pod fails, Kubernetes automatically creates a new one to replace it, and automatically mounts the appropriate PersistentVolume(s) to it.

For example, for a TimesTenClassic object called `sample`, the Operator creates a StatefulSet called `sample`, in the same Kubernetes namespace. The StatefulSet, in turn, create two Pods in the namespace, called `sample-0` and `sample-1`.

Service

A Kubernetes Service defines the set of network addresses and ports that should be exposed to applications in the cluster.

The Operator automatically creates a *headless* Service when you create the TimesTenClassic object. It automatically associates this Service with the StatefulSet. This causes Kubernetes to define entries in the Kubernetes cluster's DNS for the Pods in the StatefulSet, and to keep those DNS entries up to date.

A headless Service is used such that the DNS name/address entry for the active database is different than the DNS name/address entry for the standby database. This enables incoming client connections to be routed to the database that is active. For more information on a headless Service, see:

`https://kubernetes.io/docs/concepts/services-networking/service/#headless-services/`

For a TimesTenClassic object called `sample`, a headless Service called `sample` is also created in the same Kubernetes namespace. This results in entries in the cluster's DNS for `sample-0.sample.`*`namespace`*`.svc.cluster.local` and `sample-1.sample.`*`namespace`*`.svc.cluster.local`.

Secret

The TimesTen Operator creates a Secret to inject an SSL certificate into the TimesTen containers. This secures the communication between the TimesTen Operator and the TimesTen Agent.

Pods

The StatefulSet creates two pods. Each Pod contains two containers:

- The `tt` container. This TimesTen container is always present in the Pods. It executes the TimesTen agent and runs TimesTen.
- The `daemonlog` container: This container copies the contents of the TimesTen `ttmesg.log` file to `stdout`, resulting in Kubernetes logging the file. This enables the daemon log of the TimesTen instances to be recorded by the Kubernetes infrastructure.

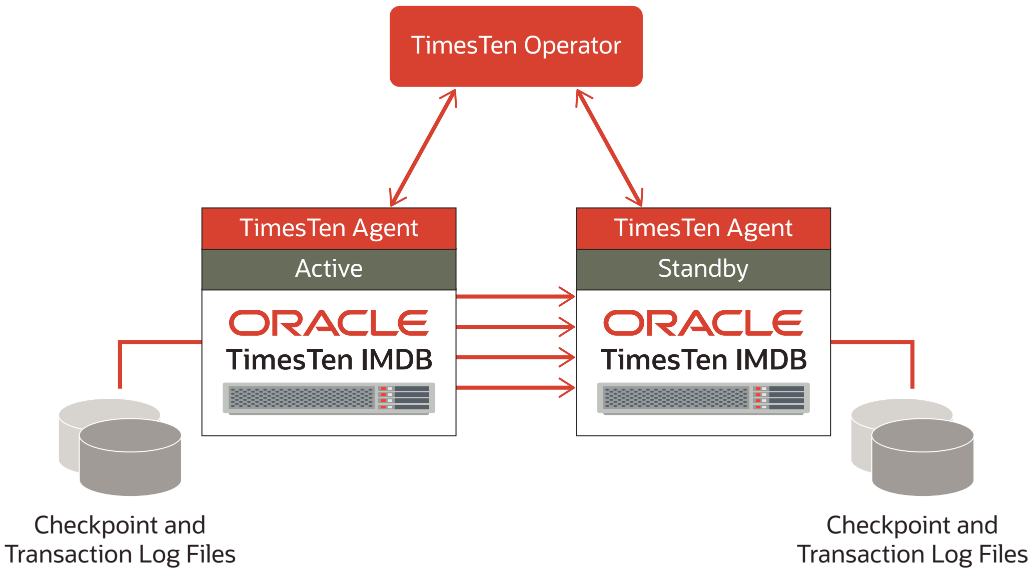
Events

The Operator creates a Kubernetes event whenever important changes occur.

About the TimesTen containers and the TimesTen agent

After the objects are created, the TimesTen containers run a script that configures and starts the TimesTen agent. The Operator communicates with the TimesTen agent running in each Pod, in order to configure, manage, and monitor TimesTen in that Pod. The agent provides an HTTPS endpoint in the Pod that the Operator uses to query and control the `tt` container in the Pod. If the TimesTen agent fails, the `tt` container automatically terminates and is re-spawned by Kubernetes. Figure 1-3 illustrates the two way communication between the Operator and the TimesTen agent.

Figure 1-3 The Operator and the TimesTen agent



The TimesTen agent starts TimesTen and thus runs as the instance administrator user. It has full control over TimesTen.

Simple deployment

The TimesTen Operator is designed for simple deployment of your active standby pairs of TimesTen Classic databases and for automated failure detection and recovery. For example,

- You decide you want to deploy a new replicated pair of TimesTen databases.
- You decide the attributes of those databases.
- You create the configuration files for those attributes.
- You use the `kubectl create` command to create a TimesTenClassic object to represent the replicated pair.
- You use the `kubectl get` and `kubectl describe` commands to observe the provisioning of the active standby pair.
- Applications that run in other Pods use TimesTen's standard client/server drivers to access TimesTen databases.

You do not have to monitor the TimesTen databases continually, configure replication, perform failover, or re-duplicate a database after failure. The TimesTen Operator performs all these functions and works to keep the databases up and running with minimal effort on your part.

About deploying a TimesTen Scaleout grid and its database

When you create a TimesTenScaleout object in the Kubernetes cluster, the process to create and configure the TimesTen Scaleout grid and its associated database begins. Kubernetes informs the TimesTen Operator that a TimesTenScaleout object has been created. The Operator begins the process of creating several Kubernetes objects that are required to deploy the grid.

Topics:

- StatefulSets
- Services
- Secret
- Persistent Volume Claims and Pods
- Password-less ssh
- Quick rollout

StatefulSets

The TimesTen Operator creates the following StatefulSets:

- One StatefulSet that provides the management instance for the grid. The underlying Pod for this management instance is also created.
- One StatefulSet that provides one or more ZooKeeper instances. There is one StatefulSet replica for each Zookeeper instance. For example, if there are three ZooKeeper instances, there is one StatefulSet and this one StatefulSet has three replicas.
- One or more additional StatefulSets, the number of which is determined by your K-Safety (`k`) value. For example, a `k` value of `2` means that there are two copies of your TimesTen database. One copy of your database resides in data space one, and the second copy in data space two. In the Kubernetes environment, the TimesTen Operator creates `k` StatefulSets. Using the previous example, if `k` is set to `2`, the Operator creates two StatefulSets. Each of the `k` StatefulSets provides Pods to implement a single data space in the resultant grid.

  The number of replicas for each StatefulSet is determined by the number of replica sets you want in your grid. A replica set contains `k` elements, where each element in the replica set is an exact copy of the other elements in the replica set. For example, if you want three replica sets in your grid, each StatefulSet has three replicas. Each replica contains a Pod for one data instance. Therefore, in this example, one StatefulSet has three Pods each of which contain one data instance. The second StatefulSet (assuming `k=2`) also has three replicas and each replica also contains a Pod for one data instance. Therefore, the second StatefulSet has three Pods each of which contain one data instance.

  In summary, in a case where `k` is set to `2` and the number of replica sets is `3`, the Operator creates two StatefulSets, each with three replicas. Each StatefulSet provides the Pods to implement a single data space. There are a total of six Pods created for the six data instances.

Services

The TimesTen Operator creates the following headless Services:

- One headless Service that provides DNS names for the Pods that contain the management and data instances. This Service enables client/server access to the Pods using the TimesTen client/server port `6625`.

- One headless Service that provides DNS names for the Pods that contain the ZooKeeper instances. This Service enables access to the Zookeeper internal ports `2888` and `3888`, as well as the external port `2181`.

Secret

The TimesTen Operator creates a Secret to inject an SSL certificate into the TimesTen containers. This secures the communication between the TimesTen Operator and the TimesTen Agent.

Persistent Volume Claims and Pods

The TimesTen Operator creates Persistent Volume Claims (PVCs) for the TimesTen and ZooKeeper Pods. These PVCs cause persistent volumes to be allocated by Kubernetes and attached to the TimesTen and ZooKeeper Pods. See Persistent storage.

Each Pod that runs a ZooKeeper instance consists of a single container called `zookeeper`.

Each Pod that runs TimesTen consists of at least two containers:

- `tt` container: Runs the TimesTen Agent and TimesTen.
- `daemonlog` container: Writes the TimesTen daemonlog to stdout. This enables the daemon log of the TimesTen instances to be recorded by the Kubernetes infrastructure.

As the `tt` containers in the TimesTen Pods start, the Operator assembles them into a working grid.

Password-less ssh

A TimesTen Scaleout grid relies on password-less ssh among the instances of the grid. The TimesTen Operator automatically configures password-less ssh among the `tt` containers in the grid in your Kubernetes environment. There is no intervention that you need to do.

Quick rollout

The TimesTen Kubernetes Operator provides the functionality for deploying TimesTen Scaleout grids and their associated databases. The TimesTen Operator rolls out the grid quickly and proficiently.

The Operator creates the StatefulSets and Services that are required to deploy the grid. It creates the ZooKeeper Pods. When the TimesTen Agent in the management Pod starts up, the Operator instructs the agent to create a TimesTen instance and grid. The Operator waits for all of the TimesTen agents in all of the Pods to start. Once all of the agents have started, the Operator instructs the agent in the management instance to create the hosts, the installations, and the instances in the grid's model for the data instances in the grid. The `DbDef` is created as are any direct and client/server connectables. The model is applied and the data instances of the grid are created.

The Operator instructs the management instance to create the database and to create the initial distribution map. The Operator then instructs the TimesTen agent in one data instance to run the TimesTen `ttIsql` utility to create initial database users and objects.

The grid and databases are created. The TimesTen agent opens the database for connections.

The grid rollout is quick and proficient with little or no intervention from you. Once the grid is rolled out, the TimesTen Operator does not manage the grid or database or perform maintenance operations.

Here is additional information:

- For information about TimesTen Scaleout: See Overview of TimesTen Scaleout in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
- For information about deploying a TimesTen Scaleout grid and database in Kubernetes: See Deploying your TimesTen Grid in this book.

2 Setting Up the Environment

This chapter describes the process for setting up the TimesTen Operator.

Topics:

Prerequisites

Review and complete the following prerequisites:

- Ensure you have a working Kubernetes cluster.
  - Your cluster must provide a *StorageClass* that can be used to request Persistent Volumes. You must know the name of this storage class. For example, in OKE, you can use the `oci` storage class. For more information on Storage Classes, see:

    `https://kubernetes.io/docs/concepts/storage/storage-classes/`
  - The nodes in your cluster must have their clocks synchronized through NTP, or equivalent.
- Ensure you have a development host to access the Kubernetes cluster. This development host must reside outside the Kubernetes cluster, and you must be able to access and to control the Kubernetes cluster from this host. On it, you must install:
  - The `kubectl` command line tool: You use the `kubectl` command line tool to control and mange the Kubernetes cluster.
  - The `docker` command line tool: You use the `docker` command line tool when working with the TimesTen container image. For example, you use the `docker` command line tool to build the TimesTen container image and to push it to your container registry.

Note:

For a list of supported Kubernetes releases, see the *Oracle TimesTen In-Memory Database Release Notes*.

About TimesTen container images

The TimesTen Operator requires a TimesTen container image. This image is used when you deploy the Operator and when you create your TimesTenClassic and TimesTenScaleout objects in your namespace.

The TimesTen container image uses the Oracle Linux base image. Oracle Java is installed into the TimesTen image using JDK script friendly URLs and Dockerfile techniques.

You have two options for obtaining and then using the TimesTen container image:

- **Option 1**: Use a TimesTen container image located at `container-registry.oracle.com/timesten`. These images contain TimesTen and its prerequisites. It has everything that is needed to run the TimesTen Operator and to run TimesTen containers in the TimesTen Pods created by the Operator. Before using this image, you must accept the *Oracle Standard Terms and Restrictions* for the image.

  The predefined values in this image are as follows. You cannot change these values:

  – `timesten`: The name of the Linux user and the Linux group that is created in the container image. The Linux user is the user who runs TimesTen. The Linux group is the TimesTen users group.

  – `3429`: The numeric UID and GID of the `timesten` user and the TimesTen users group.

- **Option 2**: Build the TimesTen container image: The TimesTen distribution contains the Operator distribution. The Operator distribution provides a Dockerfile that lets you build the TimesTen container image. The base image used in this Dockerfile is `container-registry.oracle.com/os/oraclelinux:8`.

  The Dockerfile supports a number of ARGs that let you override various attributes of the TimesTen container image. For example, you can specify a custom TimesTen user and TimesTen users group to run TimesTen rather than using the default `timesten` user and the default `timesten` group. You can also specify which TimesTen release to embed in the TimesTen container image. For the supported Dockerfile ARGs, see Dockerfile ARGs.

Let's walk through the procedures for Option 1 and Option 2. It is your choice as to which option you use.

Option 1: Use the official TimesTen container images

This option uses the official TimesTen container image located at `container-registry.oracle.com/timesten`. This image contains TimesTen and its prerequisites. It has everything that is needed to run the TimesTen Operator and to run TimesTen containers in the TimesTen Pods created by the Operator.

Complete the following tasks in the following sections:

- Accept the Oracle TimesTen license agreement
- Obtain the Operator manifest files from the official TimesTen image

Accept the Oracle TimesTen license agreement

You need to complete the steps in this section one time. If you have previously accepted the Oracle TimesTen license agreement and created your Kubernetes Secret, proceed to Obtain the Operator manifest files from the official TimesTen image.

The official TimesTen container images are located on the Oracle Container Registry at https://container-registry.oracle.com/. To use a licensed Oracle software image such as TimesTen, you must log into the Oracle Container Registry web interface and accept the *Oracle Standard Terms and Restrictions* for the image. After you complete the following steps, you can reference the official TimesTen container image in your YAML manifest files.

Note:

There are a list of TimesTen images available at `container-registry.oracle.com/timesten`. Choose the image you want to use. The examples in this book use various images. An example of a TimesTen container image is `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` image.

1. In a web browser, using your Oracle Account, log into the Oracle Container Registry located at https://container-registry.oracle.com/
2. On the **Oracle Container Registry** page, in the **Browse Containers** section, click **TimesTen**.
3. On the **TimesTen Repositories** page, in the **Repository** column, choose **timesten**.
4. On the **Official container images for the Oracle TimesTen In-Memory Database** page, to the right of the **Quick Reference Description**, locate the **Select Language** drop down list. In the **Select Language** drop down list, choose **your language.** Then, review the text before the **Continue** button and click **Continue**.

   The text that is displayed before the **Continue** button is similar to the following: "You must agree to and accept the *Oracle Standard Terms and Restrictions* prior to downloading from the Oracle Container Registry. Please read the license agreement on the following page carefully."
5. On the **Oracle Standard Terms and Restrictions** page, review the information on the page, then at the bottom of the page, click **Accept**.

   The **Official container images for the Oracle TimesTen In-Memory Database** page displays for a second time. To the right of the **Quick Reference Description**, look for a **green check mark** with text similar to the following: "You last accepted the Oracle Standard Terms and Restrictions on 01/08/2023 at 01:28 PM Coordinated Universal Time (UTC)."
6. On your development host, use the `docker login` command to authenticate against the Oracle Container Registry, using the same Oracle Account you used to log into the Oracle Container Registry web interface.

   ```
   docker login container-registry.oracle.com
   ```

   You are prompted to enter your Oracle Account and password.

   The text is similar to the following:

   ```
   Username (oracleuser@oracle.com):
   Password:
   WARNING! Your password will be stored unencrypted in /home/
   oracleuser/.docker/config.json.
   Configure a credential helper to remove this warning. See
   https://docs.docker.com/engine/reference/commandline/login/#credentials-
   ```

```
store

Login Succeeded
```

7. Create the image pull secret for `container-registry.oracle.com`.

   ```
   kubectl create secret generic sekret \
   --from-file=.dockerconfigjson=$HOME/.docker/config.json \
   --type=kubernetes.io/dockerconfigjson
   ```

   Save the name of the image pull secret. You need it later when deploying your TimesTenClassic or TimesTenScaleout objects. See Defining and creating the TimesTenClassic object and Define and deploy the TimesTenScaleout object.

   For more information about creating a Kubernetes Secret, see .

Pulling Images from Registry during DeploymentYou have successfully logged into the Oracle Container Registry and accepted the *Oracle Standard Terms and Conditions* for the official TimesTen container image. You can now reference the TimesTen container image in your YAML manifest files. Proceed to Obtain the Operator manifest files from the official TimesTen image.

Obtain the Operator manifest files from the official TimesTen image

The Operator manifest files are included within the official TimesTen container image. The official TimesTen container images are located on `container-registry.oracle.com/timesten`.

The examples in this section use the `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` image.

Complete these steps to obtain the Operator manifest files located within the `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` container image.

1. On your development host, from the directory of your choice, create the subdirectories for the Operator files. This example creates the *`kube_files`* and *`kube_files/deploy`* subdirectories and changes the default directory to *`kube_files/deploy`*.

   ```
   mkdir -p kube_files
   cd kube_files
   mkdir deploy
   cd deploy
   ```

2. Create a new container from the `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` image, supplying a name for the new container. In this example, the name of the container is `ttoper`.

   ```
   docker create --name ttoper container-registry.oracle.com/timesten/
   timesten:22.1.1.9.0
   ```

The output is similar to the following.

```
Unable to find image 'container-registry.oracle.com/timesten/
timesten:22.1.1.9.0' locally
Trying to pull repository container-registry.oracle.com/timesten/
timesten:22.1.1.9.0 ...
20221215: Pulling from container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
...
Digest:
sha256:bb58e32d2e08a66c55fb09afd8958feb91300134fc7a019bcb39a241f48fd995
Status: Downloaded newer image for container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
d03a6e534e8bcce435b86795e8b3082487e5f18af64a522ebfbabff3baec321a
```

3. Copy the Operator files from the `ttoper` container to the recently created `kube_files/deploy` directory.

   ```
   docker cp ttoper:/timesten/operator/deploy/crd.yaml .
   docker cp ttoper:/timesten/operator/deploy/operator.yaml .
   docker cp ttoper:/timesten/operator/deploy/service_account.yaml .
   ```

   Review the files.

   ```
   ls
   crd.yaml  operator.yaml  service_account.yaml
   ```

4. Remove the `ttoper` container.

   ```
   docker rm ttoper
   ```

   The output is the following.

   ```
   ttoper
   ```

5. Remove the TimesTen container image.

   ```
   docker image rm container-registry.oracle.com/timesten/timesten:22.1.1.9.0
   ```

   The output is similar to the following.

   ```
   Untagged: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
   ...
   ```

You have successfully obtained the Operator manifest files. Proceed to Deploy the TimesTen Operator to complete the Operator set up process.

Option 2: Build the TimesTen container image

The following example assumes you want to build a TimesTen container image from a TimesTen distribution. If you instead want to use the TimesTen container image located at `container-registry.oracle.com/timesten`, see Accept the Oracle TimesTen license agreement.

To build the TimesTen container image, complete these steps::

- Unpack the TimesTen and the TimesTen Kubernetes Operator distributions
- Copy the TimesTen distribution
- Choose how to build

Unpack the TimesTen and the TimesTen Kubernetes Operator distributions

To unpack the TimesTen distribution and the TimesTen Kubernetes Operator distribution, complete the following steps:

1. On your development host, from the directory of your choice:
   - Create one subdirectory for the TimesTen distribution. This example creates the *`installation_dir`* subdirectory.
   - Create a second subdirectory for the TimesTen Operator distribution. This example creates the *`kube_files`* subdirectory.

   ```
   mkdir -p installation_dir
   mkdir -p kube_files
   ```

2. Change to the TimesTen distribution subdirectory.

   ```
   cd installation_dir
   ```

3. Copy the TimesTen distribution that you previously downloaded into this TimesTen distribution directory.

   ```
   cp download_location/timesten221190.server.linux8664.zip .
   ```

4. Unpack the TimesTen distribution.

   ```
   unzip timesten221190.server.linux8664.zip
   ```

   The output is similar to the following:

   ```
   Archive:  timesten221190.server.linux8664.zip
      creating: tt22.1.1.9.0/
      ...
      creating: tt22.1.1.9.0/kubernetes/
    extracting: tt22.1.1.9.0/kubernetes/operator.zip
      ...
   ```

   The TimesTen Operator distribution is *`installation_dir`*`/tt22.1.1.9.0/kubernetes/operator.zip`.

   Do not delete the TimesTen distribution (*`installation_dir`*`/tt22.1.1.9.0`, in this example). You need it later.

5. Change to the TimesTen Operator subdirectory you created in a previous step. (In this example, *kube_files*.)

   ```
   cd kube_files
   ```

6. Unpack the TimesTen Operator distribution.

   ```
   unzip installation_dir/tt22.1.1.9.0/kubernetes/operator.zip
   ```

   The output is similar to the following:

   ```
   Archive:  installation_dir/tt22.1.1.9.0/kubernetes/operator.zip
      creating: operator/
     ...
      creating: deploy/
     inflating: deploy/service_account.yaml
     inflating: deploy/crd.yaml
     inflating: deploy/operator.yaml
     inflating: README.md
      creating: image/
     inflating: image/Dockerfile
   ```

   The TimesTen Operator distribution format includes the following directories:

   - `deploy`: Contains the YAML manifest files that are used to create the Kubernetes service account, role, and CRD. The Kubernetes service account, role, and CRD are required by the TimesTen Operator. The `deploy` directory also contains the YAML manifest file that you use to deploy the Operator in the Kubernetes cluster. These files are discussed in Create the service account and the CRD and Deploy the TimesTen Operator.
   - `operator`: Contains the `timesten-operator` Operator binary. Also contains the TimesTen Agent and the support files the Agent uses at runtime. You do not need to use or to make any modifications to the files in this directory.
   - `image`: Contains the Dockerfile that is used to create the TimesTen container image. If you are building the TimesTen container image, you use this Dockerfile.

   Make a note of these directories. You need to work with files in the `deploy` and the `image` directories in a later step.

You have successfully unpacked the TimesTen and TimesTen Operator distributions.

Copy the TimesTen distribution

The files that you need to build the TimesTen container image are provided in the unzipped TimesTen Operator distribution. In this example, the directory that contains the unzipped TimesTen Operator distribution is *kube_files*. See Unpack the TimesTen and the TimesTen Kubernetes Operator distributions.

To build the TimesTen container image, complete the following steps:

1. On your development host, change to the `image` directory of the unzipped TimesTen Operator distribution (*`kube_files`*`/image`, in this example).

   ```
   cd kube_files/image
   ```

2. Copy the TimesTen distribution into the directory. In a previous example, you created the *`installation_dir`* directory. This directory contains the TimesTen distribution.

   ```
   cp installation_dir/timesten221190.server.linux8664.zip .
   ```

   For information on *`installation_dir`*, see Unpack the TimesTen and the TimesTen Kubernetes Operator distributions.

3. Create the image pull secret for `container-registry.oracle.com`.

   ```
   kubectl create secret generic sekret \
   --from-file=.dockerconfigjson=$HOME/.docker/config.json \
   --type=kubernetes.io/dockerconfigjson
   ```

   Save the name of the image pull secret. You need it later when deploying your TimesTenClassic or TimesTenScaleout objects. See Defining and creating the TimesTenClassic object and Define and deploy the TimesTenScaleout object.

   For more information about creating a Kubernetes Secret, see Pulling Images from Registry during Deployment.

You have successfully copied the TimesTen distribution to the *`kube_files`*`/image` directory and created the image pull secret.

Choose how to build

You can choose to build the TimesTen image with the defaults specified in Dockerfile ARGs or you can choose to customize the TimesTen container image by overriding the defaults specified in Dockerfile ARGs.

Choose one option.

- Option 2a: Build with defaults
- Option 2b: Build with customizations

Option 2a: Build with defaults

The following example assumes you want to build the TimesTen container image with the defaults specified in Dockerfile ARGs.

1. On your development host, change to the *`kube_files`*`/image` directory (if not already in this directory).
2. Use the `docker build` command to build the TimesTen image. You must specify the following Dockerfile ARGs on the command line:
   - `TT_DISTRO`: The name of the file containing the TimesTen distribution (`timesten221190.server.linux8664.zip`, in this example).

- `TT_RELEASE`: The name of the TimesTen release in dotted format (`22.1.1.9.0`, in this example).

```
docker build -t tt221190image:1 \
--build-arg TT_DISTRO=timesten221190.server.linux8664.zip \
--build-arg TT_RELEASE=22.1.1.9.0 .
```

The output is similar to the following:

```
Sending build context to Docker daemon  461.6MB
Step 1/35 : ARG TT_BASE=container-registry.oracle.com/os/oraclelinux:8
Step 2/35 : ARG UNZIP_BASE=container-registry.oracle.com/os/oraclelinux:8
Step 3/35 : FROM $UNZIP_BASE as unzipper
...
Successfully built a0d9e74ad31f
Successfully tagged tt221190image:1
```

3. Tag the TimesTen container image. Replace the following:
   - `tt221190image:1` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

```
docker tag tt221190image:1 phx.ocir.io/youraccount/tt221190:1
```

4. Push the TimesTen container image to your registry. Replace the following:
   - `phx.ocir.io/youraccount` with the location of your image registry.
   - `tt221190image:1` with the name you chose previously.

```
docker push phx.ocir.io/youraccount/tt221190image:1
```

The output is similar to the following:

```
The push refers to repository [phx.ocir.io/youraccount/tt221190image]
5f494d8654f2: Pushed
cf1cdb416e24: Pushed
511cc991c8e6: Pushed
a9abe4002691: Pushed
8894990e5ffe: Pushed
c94f317a0997: Pushed
7774e1cbdcf4: Pushed
d8c569c85182: Pushed
1: digest:
sha256:8d280bd65059f089a9ca0f2131c9f2928f71f79b0708ed6920fd4df1ea7cfd68
size: 2007
```

Note:

To reduce the size of the final TimesTen container image, the Dockerfile uses a multi-stage build. This results in a dangling image left behind. To locate dangling images, use the `docker` command with the `-f` filter flag with a value of `dangling=true`. Once you locate the dangling image, you can use the `docker image prune` command to remove it. For example,

```
docker images -f dangling=true
docker image prune
```

You have successfully built the TimesTen container image. It is pushed to your image registry. Proceed to Deploy the TimesTen Operator to complete the set up process.

Option 2b: Build with customizations

The following example assumes you want to customize the TimesTen container image by overriding the defaults specified in Dockerfile ARGs. If you instead want to use these defaults, see Option 2a: Build with defaults.

This example creates a custom TimesTen user and a custom TimesTen users group to run TimesTen in the TimesTen containers.

1. On your development host, change to the *`kube_files`*`/image` directory (if not already in this directory).
2. Use the `docker build` command to build the TimesTen image. To customize the image with a custom user and a custom TimesTen users group, specify the following Dockerfile ARGs on the command line:
   - `TT_USER`: The name of the Linux operating system user created in the container image. This is the user that runs TimesTen. In this example, the name of the user is `customuser`.
   - `TT_UID`: The numeric UID of `$TT_USER`. In this example, the UID is `9876`.
   - `TT_GROUP`: The name of the Linux group created in the container image. This is the name of the TimesTen users group. In this example, the name of the group is `customgroup`.
   - `TT_GID`: The numeric GID of `$TT_GROUP`. In this example, the GID is `9876`.

   In addition, specify these required Dockerfile ARGs:
   - `TT_DISTRO`: The name of the file containing the TimesTen distribution (`timesten221190.server.linux8664.zip`, in this example).
   - `TT_RELEASE`: The name of the TimesTen release in dotted format (`22.1.1.9.0`, in this example).

   ```
   docker build -t tt221190customimage:1 \
   --build-arg TT_USER=customuser --build-arg TT_UID=9876 \
   --build-arg TT_GROUP=customgroup --build-arg TT_GID=9876 \
   ```

```
--build-arg TT_DISTRO=timesten221190.server.linux8664.zip \
--build-arg TT_RELEASE=22.1.1.9.0 .
```

The output is similar to the following:

```
Sending build context to Docker daemon  461.6MB
Step 1/35 : ARG TT_BASE=container-registry.oracle.com/os/oraclelinux:8
Step 2/35 : ARG UNZIP_BASE=container-registry.oracle.com/os/oraclelinux:8
Step 3/35 : FROM $UNZIP_BASE as unzipper
...
Successfully built 49101e32b5d5
Successfully tagged tt221190customimage:1
```

3. Tag the TimesTen container image. Replace the following:
   - `tt221190customimage:1` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

   ```
   docker tag tt221190customimage:1 phx.ocir.io/youraccount/
   tt221190customimage:1
   ```

4. Push the TimesTen container image to your registry. Replace the following:
   - `phx.ocir.io/youraccount` with the location of your image registry.
   - `tt221190customimage:1` with the name you chose previously.

   ```
   docker push phx.ocir.io/youraccount/tt221190customimage:1
   ```

   The output is similar to the following:

   ```
   The push refers to repository [phx.ocir.io/youraccount/
   tt221190customimage]
   3b5c7ba355aa: Pushed
   30047b487811: Pushed
   aa10ef33fac5: Pushed
   7290609fca00: Pushed
   676cb8be497d: Pushed
   57a50dacb378: Pushed
   d59ca9ca76e7: Pushed
   d8c569c85182: Layer already exists
   1: digest:
   sha256:616cbbb94f1a8b003a45865b51d401bc5f2b48d4f99bbe7584bcbb6304d6b75b
   size: 2007
   ```

Note:

To reduce the size of the final TimesTen container image, the Dockerfile uses a multi-stage build. This results in a dangling image left behind. To locate dangling images, use the `docker` command with the `-f` filter flag with a value of `dangling=true`. Once you locate the dangling image, you can use the `docker image prune` command to remove it. For example,

```
docker images -f dangling=true
docker image prune
```

You have successfully built the TimesTen container image. It is pushed to your image registry. Proceed to Deploy the TimesTen Operator to complete the set up process.

Deploy the TimesTen Operator

The information in these sections shows you how to customize and deploy the TimesTen Operator and how to verify that the Operator is running in your Kubernetes cluster.

- Create the service account and the CRD
- Customize the Operator
- Verify that the Operator is running

Create the service account and the CRD

The TimesTen Operator requires a Kubernetes *service account* in order to run properly. This service account requires permissions and privileges in your namespace. These permissions and privileges are granted through a *role*. The role is assigned to the service account through a role binding.

In addition, the Operator requires a CustomResourceDefinition (CRD). This CRD defines the TimesTenClassic and the TimesTenScaleout object types in the Kubernetes cluster.

When you unpacked the TimesTen Operator distribution, one of the directories that was created was the `/deploy` directory. This directory contains the following YAML manifest files:

- `service_account.yaml`: Defines the role, the role binding, and the service account for the Operator.
- `crd.yaml`: Defines the TimesTenClassic and the TimesTenScaleout object types.

The following example creates the Operator's service account and the CRD in the Kubernetes cluster:

1. On your development host, change to the *`kube_files`*`/deploy` directory.

   ```
   cd kube_files/deploy
   ```

2. Create the service account.

   ```
   kubectl create -f service_account.yaml
   ```

   The output is the following:

   ```
   role.rbac.authorization.k8s.io/timesten-operator created
   serviceaccount/timesten-operator created
   rolebinding.rbac.authorization.k8s.io/timesten-operator created
   ```

3. Create the CRD.

   ```
   kubectl create -f crd.yaml
   ```

   The output is the following:

   ```
   customresourcedefinition.apiextensions.k8s.io/
   timestenclassics.timesten.oracle.com created
   timestenscaleouts.timesten.oracle.com created
   ```

You have successfully created the Operator's service account and CRD in the Kubernetes cluster.

Customize the Operator

You can customize the TimesTen Operator by specifying the number of replicas (copies) of the Operator that you want running in your Kubernetes cluster. You also must provide the image pull secret, and the location and name of the TimesTen container image for the Operator.

1. On your development host, change to the `*kube_files*/deploy` directory (if not already in this directory).

   ```
   cd kube_files/deploy
   ```

2. Use a text editor to modify the `operator.yaml` file, making changes to these fields:

   - `replicas: 1`

     Replace `1` with the number of copies of the Operator to run in the Kubernetes cluster. `1` is acceptable for development and testing. However, you can specify more than one replica for high availability purposes.

   - Replace `sekret` with the name of the image pull secret that Kubernetes uses to pull images from your registry.

   For the TimesTen container image:

   - Replace `container-registry.oracle.com/timesten` with the location of your image registry. You determined this location in a previous step. See About TimesTen container images. This example uses `container-registry.oracle.com/timesten`.

- Replace `timesten:22.1.1.9.0` with the name of the TimesTen container image. You determined this name in a previous step. See About TimesTen container images. This example uses `timesten:22.1.1.9.0`.

```
vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timesten-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timesten-operator
  template:
    metadata:
      labels:
        name: timesten-operator
    spec:
      serviceAccountName: timesten-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timesten-operator
          image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
          command:
          - /timesten/operator/operator/timesten-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "timesten-operator"
          securityContext:
              runAsNonRoot: true
              privileged: false
              allowPrivilegeEscalation: false
              capabilities:
                  drop:
                    - all
```

3. Deploy the Operator in the namespace of your Kubernetes cluster.

```
% kubectl create -f operator.yaml
deployment.apps/timesten-operator created
```

You have successfully deployed the Operator. The Operator should now be running.

Verify that the Operator is running

Use the `kubectl get pods` command to verify that the Operator is running. If the `STATUS` field has a value of `Running`, the Operator is running.

```
kubectl get pods
```

The output is similar to the following:

```
NAME                                 READY   STATUS    RESTARTS   AGE
timesten-operator-5c7558cd75-dfbd7   1/1     Running   0          13s
```

You have successfully verified that the Operator is running in your Kubernetes cluster.

3

Using Configuration Metadata

This chapter gives an overview of the configuration metadata that is supported in the TimesTen Operator. It also discusses the Kubernetes facilities that you can use to get the configuration metadata into your TimesTen containers. The chapter then discusses additional configuration options. There are examples throughout.

Topics:

Overview of the configuration metadata and the Kubernetes facilities

Configuration metadata lets you define the attributes of your TimesTen database and how that database is to interact with other applications and components. The TimesTen Operator supports several metadata files that contain the configuration metadata. Each metadata file has a specific name. You use a text editor to create the metadata file with the specific name and then add the appropriate metadata to it. For example, the TimesTen Operator supports the `db.ini` metadata file. You use your editor to create the `db.ini` file and in it you define attributes for you database.

There is configuration metadata for the following TimesTen Operator deployments:

- Active standby pair of TimesTen databases in TimesTen Classic
- Active standby pair of TimesTen databases with TimesTen Cache in TimesTen Classic
- TimesTen Scaleout grid and the database within the grid

Kubernetes supports various facilities that places the metadata files into the `/ttconfig` directory of the TimesTen containers. See Populating the /ttconfig directory.

List of the configuration metadata

Table 3-1 lists the metadata files that are supported by the TimesTen Operator. The table provides a description for each of the metadata files and indicates if the metadata file is supported in TimesTen Classic, in TimesTen Scaleout, or in both.

Table 3-1 TimesTen Operator metadata files

Name	Description	TimesTen Classic support	TimesTen Scaleout support
adminUser	Defines an initial user in the database and assigns this user `ADMIN` privileges. Optional.	Yes	Yes
cachegroups.sql	Defines the cache groups in the database. This file is specific to TimesTen Cache. Required if using TimesTen Cache.	Yes	No
cacheUser	Defines the cache administration user in the database. This file is specific to TimesTen Cache. Required if using TimesTen Cache.	Yes	No
csWallet	Defines the credentials that are used for Transport Layer Security (TLS) encryption of client/server communications. Required if using TLS.	Yes	Yes
db.ini	Defines the connection attributes of the database. See List of attributes in the *Oracle TimesTen In-Memory Database Reference*. Required if using TimesTen Cache. Otherwise, optional.	Yes	Yes
epilog.sql	Performs operations after the replication scheme is created. Optional.	Yes	No
replicationWallet	Defines the credentials that are used for Transport Layer Security (TLS) encryption of replication traffic between the TimesTen databases. Required if using TLS.	Yes	No
schema.sql	Defines an initial schema for the database. Optional.	Yes	Yes
sqlnet.ora	Defines how client applications communicate with an Oracle database. This file is specific to TimesTen Cache. Optional.	Yes	No
tnsnames.ora	Defines the Oracle Database service that TimesTen Cache uses to connect to Oracle Database. Required if using TimesTen Cache.	Yes	No
*.connect	Defines one or more direct connectables for the database in TimesTen Scaleout. Optional.	No	Yes
*.csconnect	Defines one or more client/server connectables for the database in TimesTen Scaleout. Optional.	No	Yes

Configuration metadata details

Metadata files let you specify the attributes and the metadata for your TimesTen database. After you create these files, and you choose a facility to get these files in your TimesTen containers, TimesTen accesses them to determine the attributes and the metadata that is specific to your database.

adminUser

The `adminUser` file creates an initial user with `ADMIN` privileges in the TimesTen database. If you provide this file, this user is created after the database is created. This file must contain one line of the form:

```
user/password
```

cachegroups.sql

The `cachegroups.sql` file contains the create cache group definitions and the cache group operations for your database. You can specify the following cache group definitions and cache operations in this file:

- (Required): `CREATE CACHE GROUP` statements to create TimesTen cache groups
- (Optional): `LOAD CACHE GROUP` statements to load data from the Oracle database into your cache groups
- (Optional): `ttOptUpdateStats` or `ttOptEstimateStats` TimesTen built-in procedures to update statistics on the cache tables

The `cachegroups.sql` file is required if you are using TimesTen Cache in your TimesTenClassic deployment. This requirement ensures cache groups are created before replication is configured. Note: The instance administrator uses the `ttIsql` utility to run the `cachegroups.sql` file.

See:

- CREATE CACHE GROUP and LOAD CACHE GROUP in the *Oracle TimesTen In-Memory Database SQL Reference*
- Cache group types in the *Oracle TimesTen In-Memory Database Cache Guide*
- ttOptUpdateStats and ttOptUpdateStats in the *Oracle TimesTen In-Memory Database Reference*

Here is an example of a `cachegroups.sql` file. The file defines two cache groups and loads data into one cache group.

```
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP writecache
FROM oratt.writetab (
  pk NUMBER NOT NULL PRIMARY KEY,
  attr VARCHAR2(40)
);

CREATE READONLY CACHE GROUP readcache
AUTOREFRESH
  INTERVAL 5 SECONDS
FROM oratt.readtab (
  keyval NUMBER NOT NULL PRIMARY KEY,
  str VARCHAR2(32)
);

LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
```

cacheUser

The `cacheUser` file lets you create the TimesTen cache manager user. This user must have the same name as the cache administration user in the Oracle database, and must already exist in the Oracle database. See Create the Oracle database users and default tablespace in the *Oracle TimesTen In-Memory Database Cache Guide*.

This file must contain one line of the form,

```
cacheUser/ttPassword/oraPassword
```

where `cacheUser` is the TimesTen cache manager user, `ttPassword` is the TimesTen password for the TimesTen `cacheUser` user, and `oraPassword` is the Oracle database password you specified when you created the `cacheUser` user in the Oracle database.

For example, assume you have created the `cacheuser2` cache administration user in the Oracle Database with password `oraclepwd`. Assume you designate this `cacheuser2` user as the TimesTen cache manager user with a TimesTen password of `ttpwd`. In this example, the `cacheUser` metadata file contains this one line:

```
cacheuser2/ttpwd/oraclepwd
```

The TimesTen Operator creates the `cacheuser2` user with the `ttpwd` in the TimesTen database. This `cacheuser2` user then serves as the cache manager user in your TimesTen database. You do not need to create this TimesTen user. The Operator does it for you.

See Create the TimesTen users in the Oracle TimesTen In-Memory Database Cache Guide.

The Operator grants privileges to the TimesTen `cacheUser` user (`cacheuser2`, in this example) that are appropriate for this user's role as the cache manager. These privileges are:

- `CREATE SESSION`
- `CACHE MANAGER`
- `CREATE ANY TABLE`
- `LOAD ANY CACHE GROUP`
- `REFRESH ANY CACHE GROUP`
- `FLUSH ANY CACHE GROUP`
- `DROP ANY CACHE GROUP`
- `ALTER ANY CACHE GROUP`
- `UNLOAD ANY CACHE GROUP`
- `SELECT ANY TABLE`
- `INSERT ANY TABLE`
- `UPDATE ANY TABLE`

- `DELETE ANY TABLE`

csWallet

In a TimesTen Client/Server environment, data is transmitted between your client applications and your TimesTen database unencrypted by default. However, you can configure TLS for Client/Server to ensure secure network communication between TimesTen clients and servers. To encrypt Client/Server traffic, specify the `/ttconfig/csWallet` file. This file contains the Oracle wallet for the server, which contains the credentials that are used for configuring TLS encryption between your TimesTen database and your Client/Server applications. The file will be available in the containers of your TimesTen databases in the directory `/tt/home/timesten/csWallet`. You can reference this directory in your `db.ini` file (by specifying the `wallet` connection attribute). See Creating TLS certificates for replication and Client/Server and Creating TLS certificates for replication and Client/Server.

The client wallet must also be available to your client applications. See Creating TLS certificates for replication and Client/Server and Configuring TLS for Client/Server.

db.ini

The `db.ini` file contains the TimesTen DSN definition for your database.

In TimesTen Classic, the `db.ini` file contains the connection attributes for your database. This file is used to to generate the `sys.odbc.ini` file for the instances. You can specify data store attributes, first connection attributes, and general connection attributes in the `db.ini` file. The name of the DSN is the name of the TimesTenClassic object. For example, if your TimesTenClassic object is called `sample`, the name of your DSN is `sample`.

In TimesTenScaleout, the `db.ini` file contains the connection attributes for each element of your database in the grid. The database definition file (dbDef) and its contents are used to create a TimesTen Scaleout database definition. The name of the database definition is the name of the TimesTenScaleout object. For example, if the name of your TimesTenScaleout object is `sample`, the name of the database definition is `sample`. The TimesTen Scaleout database is created based on the database definition. For information about creating a database and creating a database definition in TimesTen Scaleout, see Creating a database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

If you are using TimesTen Cache in your TimesTenClassic deployment, you must specify the `OracleNetServiceName` and the `DatabaseCharacterSet` connection attributes in the `db.ini` file. The `DatabaseCharacterSet` value must match the value of the database character set in the Oracle Database.

Do not specify the the `DataStore` or the `LogDir` connection attributes in the `db.ini` file. The Operator sets these attributes, placing the database files in Kubernetes Persistent Volumes.

See "List of attributes" in the *Oracle TimesTen In-Memory Database Reference* for information on the TimesTen connection attributes.

Note:

If the `/ttconfig/db.ini` file is not present in a TimesTen container, TimesTen creates a default `sys.odbc.ini` file. For this default `sys.odbc.ini`, the connection attributes are: `Permsize=200 DatabaseCharacterSet=AL32UTF8`

This example shows a sample `db.ini` file that contains various connection attributes for TimesTen databases in TimesTen Classic or TimesTen Scaleout.

```
PermSize=500
LogFileSize=1024
LogBufMB=1024
DatabaseCharacterSet=AL32UTF8
```

Here is an example that shows a sample `db.ini` file that contains the `OracleNetServiceName` for TimesTen databases that use TimesTen Cache in TimesTen Classic.

```
PermSize=500
LogFileSize=1024
LogBufMB=1024
DatabaseCharacterSet=AL32UTF8
OracleNetServiceName=OraCache
```

epilog.sql

In TimesTen Classic, the `epilog.sql` file includes operations that occur after the replication scheme has been created and the replication agent has been started. For example, if you want to create replicated bookmarks in XLA, you can include the `ttXlaBookmarkCreate` TimesTen built-in procedure in this file.

The Operator instructs the instance administrator to run the `epilog.sql` file using the `ttIsql` utility.

Here is an example of an `epilog.sql` file. The example calls the `ttXlaBookmarkCreate` TimesTen built-in procedure to create XLA bookmarks.

```
call ttXlaBookmarkCreate('mybookmark',0x01);
```

For information about replicated bookmarks, see the ttXlaBookmarkCreate TimesTen built-in procedure in the *Oracle TimesTen In-Memory Database Reference*.

replicationWallet

In TimesTen Classic, TimesTen replication transmits data between your TimesTen databases unencrypted by default. However, you can configure TLS for replication to ensure secure network communication between your replicated TimesTen databases. To do this, specify the `/ttconfig/replicationWallet` file. This file contains an Oracle wallet, which contains the credentials that are used by TimesTen replication for configuring TLS encryption between your active standby pair of TimesTen databases. See Creating TLS certificates for replication and Client/Server and Configuring TLS for replication.

If you specify this file, you must include the `replicationCipherSuite` field and optionally include the `replicationSSLMandatory` field in your TimesTenClassic object definition. See the `replicationCipherSuite` entry and the `replicationSSLMandatory` entry in TimesTenClassicSpecSpec and Configuring TLS for replication.

schema.sql

The TimesTen Operator can automatically initialize your database with schema objects, such as users, tables, and sequences. To have the Operator do this, create the `schema.sql` file.

The Operator directs the instance administrator to use the `ttIsql` utility to run the `schema.sql` file immediately after the database is created. This operation occurs before the Operator configures replication or cache in your TimesTen database.

In TimesTen Cache, one or more cache table users own the cache tables. If this cache table user is not the cache manager user, then you must specify the `schema.sql` file and in it you must include the schema user and assign the appropriate privileges to this schema user. For example, if the `oratt` schema user was created in the Oracle database, and this user is not the TimesTen cache manager user, you must create the TimesTen `oratt` user in this file. See Create the Oracle Database users.

Do not include cache definitions in this file. Instead, use the `cachegroups.sql` metadata file. See cachegroups.sql.

sqlnet.ora

The Oracle Database `sqlnet.ora` file defines the options for how client applications communicate with the Oracle Database. To use TimesTen Cache or to use tools like `ttLoadFromOracle`, define a `sqlnet.ora` file. This file describes how applications, including TimesTen, can connect to your Oracle database. Note: If you define a `sqlnet.ora` file, you must define a `tnsnames.ora` file. See tnsnames.ora.

This is an example of a `sqlnet.ora` file:

```
NAME.DIRECTORY_PATH= {TNSNAMES, EZCONNECT, HOSTNAME}
SQLNET.EXPIRE_TIME = 10
SSL_VERSION = 1.2
```

tnsnames.ora

The Oracle Database `tnsnames.ora` file defines Oracle Net Services to which applications connect. You need to use `tnsnames.ora` (and perhaps a `sqlnet.ora` file, described in sqlnet.ora) if you are using:

- TimesTen Cache
- SQL APIs, such as Pro*C, OCI, or ODPI-C
- The `ttLoadFromOracle` feature

For information about the `ttLoadFromOracle` TimesTen built-in procedure, see ttLoadFromOracle in the *Oracle TimesTen In-Memory Database Reference*.

Here is an example of a `tnsnames.ora` file:

```
OraTest =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.mynamespace.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraTest.my.sample.com)))
```

```
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.mynamespace.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.sample.com)))
```

*.connect

Files with names that end with the `.connect` extension define one or more direct connectables for direct mode access to a database in TimesTen Scaleout. You can create as many direct connectables as you like. A direct connectable specifies a set of general connection attribute settings for the database. You can choose any name for the direct connectable as long as it is a valid DSN name. The TimesTen Operator creates one direct connectable for each direct connectable file you create. The `.connect` extension denotes a direct connectable. Ensure the `.connect` extension is in lowercase.

The following example creates one direct connectable. The name of the file is `sample.connect`. The TimesTen Operator creates the `sample` direct connectable based on the contents of the `sample.connect` file.

```
ConnectionCharacterSet=AL32UTF8
```

For more information about direct connectables, see Connectable operations in the *Oracle TimesTen In-Memory Database Reference* and Connecting to a database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

*.csconnect

Files with names that end with the `.csconnect` extension define one or more client/server connectables for client/server access to a database in TimesTen Scaleout. You can create as many client/server connectables as you like. A client/server connectable specifies a set of general connection attribute settings for the database. You can choose any name for the client/server connectable as long as it is a valid DSN name. The TimesTen Operator creates one client/server connectable for each client/server connectable file you create. The `.csconnect` extension denotes a client/server connectable. Ensure the `.csconnect` extension is in lowercase.

The following example creates one client/server connectable. The name of the file is `samplecs.csconnect`. The TimesTen Operator creates the `samplecs` client/server connectable based on the contents of the `samplecs.csconnect` file.

```
TTC_Timeout=30
```

For more information about client/server connectables, see Connectable operations in the *Oracle TimesTen In-Memory Database Reference* and Connecting to a database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Populating the /ttconfig directory

You can use different methods to ensure metadata files are placed in the `/ttconfig` of the TimesTen containers. There is no requirement as to which method to use. Kubernetes provides such facilities as ConfigMaps, Secrets, and init containers for you to consider.

- Using ConfigMaps and Secrets
- Using an init container

Using ConfigMaps and Secrets

You can use one or more ConfigMaps and one or more Secrets to incorporate metadata files into the TimesTen containers. This lets you specify different TimesTen metadata for different deployments. In addition, you can use Secrets for metadata that contains sensitive data, like passwords and certificates.

The use of a ConfigMap to populate the metadata into Pods is a standard Kubernetes technique. One benefit is that you can modify the ConfigMap after it is created, which results in the immediate update of the files that are in the Pod.

Note:

TimesTen may not immediately notice and act on the changed content of the files.

When you use ConfigMaps and Secrets to hold your metadata and then reference them in the TimesTenClassic object definition, the TimesTen Operator creates a Projected Volume called `tt-config`. This `tt-config` volume contains the contents of all the ConfigMaps and all the Secrets specified in the `dbConfigMap` and the `dbSecret` fields of your TimesTenClassic or your TimesTenScaleout object. This volume is mounted as `/ttconfig` in the TimesTen containers.

Note:

You can specify one or more ConfigMaps and/or Secrets in your TimesTenClassic or TimesTenScaleout object using the `dbConfigMap` and `dbSecret` datum. The result is that these ConfigMaps and/or Secrets are mounted read-only at `/ttconfig`. Since such a mount is read-only, you cannot write into it from an init container. Alternatively, you can use an `emptydir` volume and use an init container to write files into it. However, you cannot combine ConfigMaps and Secrets with an init container. For information about using an init container, see Using an init container.

To use ConfigMaps and Secrets, follow this process:

- Decide what facilities will contain what metadata files. For example, you can use one ConfigMap for all the metadata files. Or, for example, you can use one ConfigMap for the `db.ini` metadata file and one Secret for the `adminUser` and the `schema.sql` metadata files. There is no specific requirement.
- Create the directory (or directories) that will contain the metadata files.

- Use the `kubectl create` command to create the ConfigMap and the Secrets in the Kubernetes cluster.
- Include the ConfigMaps and Secrets in your TimesTenClassic or your TimesTenScaleout object definition.

The following examples illustrate how to use ConfigMaps and Secrets for a TimesTenClassic or a TimesTenScaleout object.

- Example using one ConfigMap
- Example using one ConfigMap and one Secret
- Example using one ConfigMap for a TimesTenScaleout object

Example using one ConfigMap

This example uses one ConfigMap (called `sample`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files.

You can use this ConfigMap for a TimesTenClassic or a TimesTenScaleout object.

1. On your development host, from the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_sample` subdirectory. (The *`cm_sample`* directory is used in the remainder of this example to denote this directory.)

   ```
   mkdir -p cm_sample
   ```

2. Change to the ConfigMap directory.

   ```
   cd cm_sample
   ```

3. Create the `db.ini` file. In this `db.ini` file, define the `PermSize` and `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ```

4. Create the `adminUser` file. In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

5. Create the `schema.sql` file. In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator automatically initializes your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

6. Create the ConfigMap. The files in the `cm_sample` directory are included in the ConfigMap. These files are later available in the TimesTen containers.

   In this example:

- The name of the ConfigMap is `sample`. Replace `sample` with a name of your choosing.
- This example uses `cm_sample` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_sample` with the name of your directory.

```
kubectl create configmap sample --from-file=cm_sample
```

The output is the following:

```
configmap/sample created
```

7. Verify the contents of the ConfigMap.

```
kubectl describe configmap sample
```

The output is the following:

```
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
adminUser:
----
sampleuser/samplepw

db.ini:
----
PermSize=200
DatabaseCharacterSet=AL32UTF8

schema.sql:
----
create sequence sampleuser.s;
create table sampleuser.emp (
  id number not null primary key,
  name char(32)
);

Events:  <none>
```

8. Include the ConfigMap in the object definition. In the `dbConfigMap` field, specify the name of the your ConfigMap (`sample`, in this example).

   Note this example uses a `storageSize` of `250Gi` (suitable for a production environment). For demonstration purposes, a `storageSize` of `50Gi` is adequate.

   This is an example of using the ConfigMap for a TimesTenClassic object.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - sample
```

This is an example of using the ConfigMap for a TimesTenScaleout object.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenScaleout
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    imagePullSecret: sekret
    k: 2
    nReplicaSets: 3
    nZookeeper: 3
    dbConfigMap:
    - sample
```

The `sample` ConfigMap holds the metadata files. The `tt-config` volume contains the contents of the `sample` ConfigMap.

Example using one ConfigMap and one Secret

This example uses one ConfigMap (called `myconfig`) for the `db.ini` metadata file and one Secret (called `mysecret`) for the `adminUser` and the `schema.sql` metadata files.

You can use this ConfigMap and Secret for a TimesTenClassic or a TimesTenScaleout object.

1. On your development host, from the directory of your choice:
   - Create one empty subdirectory for the ConfigMap. This example creates the `cm_myconfig` subdirectory. (The *`cm_myconfig`* directory is used in the remainder of this example to denote this directory.) This directory will contain the `db.ini` metadata file.
   - Create a second empty subdirectory for the Secret. This example creates the `secret_mysecret` subdirectory. (The `secret_mysecret` directory is used in the remainder of this example to denote this directory.) This directory will contain the `adminUser` and the `schema.sql` metadata files.

   ```
   mkdir -p cm_myconfig
   mkdir -p secret_mysecret
   ```

2. Change to the ConfigMap directory.

   ```
   cd cm_myconfig
   ```

3. Create the `db.ini` file in this ConfigMap directory (`cm_myconf`, in this example). In this `db.ini` file, define the `PermSize` and `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ```

4. Change to the Secret directory.

   ```
   cd secret_mysecret
   ```

5. Create the `adminUser` file in this Secret directory (`secret_mysecret` in this example). In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

6. Create the `schema.sql` file in this Secret directory (`secret_mysecret` in this example). In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator automatically initializes your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

7. Create the ConfigMap. The files in the `cm_myconfig` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

   In this example:

   - The name of the ConfigMap is `myconfig`. Replace `myconfig` with a name of your choosing.
   - This example uses `cm_myconfig` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_myconfig` with the name of your directory.

   Create the ConfigMap.

   ```
   kubectl create configmap myconfig --from-file=cm_myconfig
   ```

   The output is the following:

   ```
   configmap/myconfig created
   ```

8. Verify the contents of the ConfigMap.

   ```
   kubectl describe configmap myconf
   ```

   The output is the following:

   ```
   Name:         myconfig
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   ```

```
Data
====
db.ini:
----
PermSize=200
DatabaseCharacterSet=AL32UTF8

Events:  <none>
```

9. Create the Secret. The files in the `secret_mysecret` directory are included in the Secret and, later, will be available in the TimesTen containers.

   In this example:

   - The name of the Secret is `mysecret`. Replace `mysecret` with a name of your choosing.
   - This example uses `secret_mysecret` as the directory where the files that will be copied into the Secret reside. If you use a different directory, replace `secret_mysecret` with the name of your directory.

   ```
   kubectl create secret generic mysecret --from-file=secret_mysecret
   ```

   The output is the following:

   ```
   secret/mysecret created
   ```

10. Verify the Secret. Note the contents of the `adminUser` and the `schema.sql` files are not displayed.

    ```
    kubectl describe secret mysecret
    ```

    The output is the following:

    ```
    Name:         mysecret
    Namespace:    mynamespace
    Labels:       <none>
    Annotations:  <none>

    Type:  Opaque

    Data
    ====
    adminUser:   12 bytes
    schema.sql:  98 bytes
    ```

11. Include the ConfigMap and the Secret in the object definition.

    - In the `dbConfigMap` field, specify the name of the your ConfigMap.
    - In the `dbSecret` field, specify the name of the your Secret.

    This is an example of using the ConfigMap and the Secret for a TimesTenClassic object.

    ```
    apiVersion: timesten.oracle.com/v1
    kind: TimesTenClassic
    metadata:
    ```

```
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - myconfig
    dbSecret:
    - mysecret
```

This is an example of using the ConfigMap and the Secret for a TimesTenScaleout object.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenScaleout
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    k: 2
    nReplicaSets: 3
    nZookeeper: 3
    dbConfigMap:
    - myconfig
    dbSecret:
    - mysecret
```

The `myconfig` ConfigMap and the `mysecret` Secret holds the metadata files. The `tt-config` volume contains the contents of the `myconfig` ConfigMap and the `mysecret` Secret.

Example using one ConfigMap for a TimesTenScaleout object

This example shows you how to create a metadata file for a direct connectable and a metadata file for a client/server connectable. It then shows you how to include these connectables in a ConfigMap for a TimesTenScaleout object. The ConfigMap also includes the `db.ini`, `adminUser`, and `schema.sql` metadata files.

1. On your development host, from the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_samplescaleout` subdirectory. (The *`cm_samplescaleout`* directory is used in the remainder of this example to denote this directory.)

   ```
   mkdir -p cm_samplescaleout
   ```

2. Change to the ConfigMap directory.

   ```
   cd cm_samplescaleout
   ```

3. Create the direct connectable.

   ```
   vi samplescaleout.connect

   ConnectionCharacterSet=AL32UTF8
   ```

4. Create the client/server connectable.

   ```
   vi samplecsscaleout.csconnect

   TTC_Timeout=30
   ```

5. Create the `db.ini` file. In this `db.ini` file, define the `PermSize` and `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ```

6. Create the `adminUser` file. In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

7. Create the `schema.sql` file. In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator automatically initializes your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

8. Create the ConfigMap. The files in the `cm_samplescaleout` directory are included in the ConfigMap. These files are later available in the TimesTen containers.

   In this example:

   - The name of the ConfigMap is `samplescaleout`. Replace `samplescaleout` with a name of your choosing.
   - This example uses `cm_samplescaleout` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_samplescaleout` with the name of your directory.

   ```
   kubectl create configmap samplescaleout --from-file=cm_samplescaleout
   ```

   The output is the following:

   ```
   configmap/samplescaleout created
   ```

9. Verify the contents of the ConfigMap.

   ```
   kubectl describe configmap samplescaleout
   ```

The output is the following:

```
Name:         samplescaleout
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
adminUser:
----
sampleuser/samplepw

db.ini:
----
PermSize=200
DatabaseCharacterSet=AL32UTF8

sampleconnectable.connect:
----
ConnectionCharacterSet=AL32UTF8

samplecsconnectable.csconnect:
----
TTC_Timeout=30

schema.sql:
----
create sequence sampleuser.s;
create table sampleuser.emp (
  id number not null primary key,
  name char(32)
);

Events:  <none>
```

10. Include the ConfigMap in the object definition.

    Note this example uses a `storageSize` of `250Gi` (suitable for a production environment). For demonstration purposes, a `storageSize` of `50Gi` is adequate.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenScaleout
metadata:
  name: samplescaleout
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    k: 2
    nReplicaSets: 3
    nZookeeper: 3
    dbConfigMap:
    - samplescaleout
```

Using an init container

You can use an init container to place your metadata files into the `/ttconfig` directory of the TimesTen containers. An init container lets you to create your own scripts to populate the `/ttconfig` directory. You can use an init container for TimesTenClassic and for TimesTenScaleout objects. For more information about init containers, see:

https://kubernetes.io/docs/concepts/workloads/pods/init-containers

Note:

You can specify one or more ConfigMaps and/or Secrets in your TimesTenClassic or TimesTenScaleout object using the `dbConfigMap` and `dbSecret` datum. The result is that these ConfigMaps and/or Secrets are mounted read-only at `/ttconfig`. Since such a mount is read-only, you cannot write into it from an init container. Alternatively, you can use an `emptydir` volume and use an init container to write files into it. However, you cannot combine ConfigMaps and Secrets with an init container. For information about using ConfigMaps and Secrets, see Using ConfigMaps and Secrets.

Here is an example that illustrates how to use an init container for a TimesTenClassic object. The `template` element is required. This element is applied to Pods that contain the TimesTen Classic instances. The example shows you where to specify the script that populates the `/ttconfig` directory. It also uses the `tt-config` volume name in the `volumes` field of the TimesTenClassic object. If you specify a volume with the `tt-config` name, it is automatically mounted at `/ttconfig` in your TimesTen containers.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: init1
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
  template:
    spec:
      imagePullSecrets:
      - name: sekret
      initContainers:
      - name: initclassic
        image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
        command:
        - sh
        - "-c"
        - |
          /bin/bash <<'EOF'
          Your script to populate /ttconfig goes here
          EOF
        volumeMounts:
        - name: tt-config
          mountPath: /ttconfig
      volumes:
      - name: tt-config
        emptyDir: {}
```

When using an init container for a TimesTenScaleout object, the metadata files must be placed in both the `dataTemplate` and the `mgmtTemplate` elements.

Additional configuration options

This section discusses additional configuration options. These are optional configurations for your environment:

- Persistent storage
- Additional resource specifications
- Readiness probes

Persistent storage

When you create a TimesTenClassic object, the Operator automatically creates one or two Persistent Volume Claims (PVCs) per Pod. These PVCs cause Persistent Volumes (PVs) to be allocated by Kubernetes and to be attached to the TimesTen Pods. TimesTen uses the PVs to hold the TimesTen instance and the TimesTen database. If you specify two PVCs, one PV holds the instance and the checkpoint files and the second PV holds the transaction log files.

When you create a TimesTenScaleout object, the same mechanism is used to provision persistent storage for the data instances, the management instance, and the ZooKeeper instances:

- For the data instances: The Operator creates one or two PVCs per Pod. TimesTen uses the PVs to hold the TimesTen instance and the TimesTen database. If you specify two PVCs, one PV holds the instance and the checkpoint files and the second PV holds the transaction log files.
- For the management instance: The Operator creates one PVC for the Pod that contains the management instance. The PV holds the the TimesTen management instance and the grid database.
- For the ZooKeeper instances: The Operator creates one PVC for each Pod that runs a ZooKeeper instance. The PV holds ZooKeeper's persistent data.

When you create a TimesTenClassic object, you must specify `storageClassName` and you may specify `storageSize`. These attributes determine the characteristics of the Persistent Volumes. The `storageClassName` must be one that is provided in your Kubernetes environment. For example, in Oracle Kubernetes Environment (OKE), you may use `oci`.

The default storage is `50Gi`. Use the `storageSize` attribute to request a different size. A storage size of `50Gi` may be adequate for demonstration purposes, but in production environments, you should consider greater storage.

TimesTen places the TimesTen installation, the instance, and the database in this storage. It is mounted in each container, in each Pod, as `/tt`. The TimesTen instance is located at `/tt/home/timesten/instances/instance1`.

When you create a TimesTenScaleout object, the following attributes are supported:

- Storage class name:
  - `dataStorageClassName`: Name of the storage class that is used to request persistent volumes for the elements of the TimesTen database in the grid. If not specified, the default is the value of `storageClassName`.
  - `mgmtStorageClassName`: Name of the storage class that is used to request persistent volumes for the database of the management instance. If not specified, the default is the value of `storageClassName`.
  - `zookeeperStorageClassName`: Name of the storage class that is used to request persistent volumes for ZooKeeper's persistent data. If not specified, the default is the value of `storageClassName`.
  - `storageClassName`: If the data storage class name, the management storage class name, and the zookeeper storage class name are the same, you can just specify `storageClassName`.
- Storage size:
  - `dataStorageSize`: Amount of storage to be provisioned for each element of the TimesTen database in the grid. The default is `50Gi`.
  - `mgmtStorageSize`: Amount of storage to be provisioned for the database of the management instance. The default is `50Gi`.
  - `zookeeperStorageSize`: Amount of storage to be provisioned for ZooKeeper's persistent data. The default is `50Gi`.
  - `storageSize`: If the data storage size, the management storage size, and the zookeeper storage size are the same, you can just specify `storageSize`. For example, if the `dataStorageSize` is `75Gi`, and the `mgmtStorageSize` is `75Gi`, and the `zookeeperStorageSize` is `75Gi`, you can specify `storageSize` with a value of `75Gi`. The value for `dataStorageSize`, for `mgmtStorageSize`, and for `zookeeperStorageSize` is set to the value of `storageSize`.

For the TimesTen databases (using TimesTen Classic) and for the TimesTen database (using TimesTen Scaleout):

- TimesTen best practices recommends that the transaction log files associated with a TimesTen database be located on a different storage volume than the checkpoint files for the database. This provides separate paths to storage for the checkpoint and the transaction log operations. For example, you can store the transaction log files in a high performance storage, while storing the checkpoint files in a slower storage. See "Locate checkpoint and transaction log files on separate physical device" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.
- To locate the checkpoint files and the transaction log files on a separate path of storage, provide a value for a second persistent storage that is used only for the transaction log files. Use the `logStorageSize` attribute for this and control its placement by using the `logStorageClassName` attribute. This causes a second PVC to be created for each Pod, which will then be available in each container at `/ttlog`. (This second storage volume has a `/ttlog` mount point.)

Here is an example for a TimesTenClassic object. The same example can be used for a TimesTenScaleout object:

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: slower
    storageSize: 750Gi
    logStorageClassName: faster
    logStorageSize: 200G
```

Additional resource specifications

Kubernetes supports affinity and anti-affinity settings that let applications control their placement within the Kubernetes cluster. These settings can be used to ensure all replicas do not reside on a single physical host.

You can specify affinity settings, node selectors, additional containers, tolerations, resource requirements, and other Kubernetes attributes for the TimesTen Pods and the containers within these Pods that are created by the TimesTen Operator.

In a TimesTenClassic deployment, you specify these resource specifications in the TimesTenClassic object's `.spec.template` datum. The TimesTen Operator passes this `template` to the StatefulSet. For example, when you deploy a TimesTenClassic object, the Operator configures a replicated pair of TimesTen databases that provide high availability. However, since the Operator does not control the placement of Pods, you can achieve an even greater level of high availability by controlling the placement of the TimesTen Pods. TimesTen Pods can then be available in different availability zones or are on different Kubernetes nodes. To do this, you specify the `affinity` option in the `.spec.template` datum for the TimesTenClassic object.

Similar to a TimesTenClassic deployment, you can specify the same resource specifications for a TimesTenScaleout object. The TimesTenScaleout object supports the `.spec.mgmtTemplate`, `.spec.dataTemplate` and `.spec.zookeeperTemplate` attributes. You can use these attributes to pass affinity and other settings to Kubernetes. These are of type `PodTemplateSpec`:

- `mgmtTemplate`: Applied to the Pod that contains the TimesTen Scaleout management instance. Consists of a single `PodTemplateSpec`.
- `dataTemplate`: Applied to the Pods that contain the TimesTen Scaleout data instances. Consists of an array of `PodTemplateSpec`. If specified, there must be one entry in the array for each data space in the grid (`k` entries in the array). This lets you specify a different placement for each data space. For example, you can have data space one reside in availability zone 1 and data space two reside in availability zone 2.
- `zookeeperTemplate`: Applied to the Pods that contain the ZooKeeper instances used by Scaleout. Consists of a single `PodTemplateSpec`.

For information about `PodTemplateSpec` see, `https://kubernetes.io/docs/reference/kubernetes-api/`

Here is an example of specifying the `affinity` setting for a TimesTenClassic object.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
```

```
metadata:
  name: sample
spec:
  …
  template:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            podAffinityTerm:
             labelSelector:
              matchExpressions:
               - key: "app"
                 operator: In
                 values:
                  - ds1
             topologyKey: "kubernetes.io/hostname"
```

Readiness probes

Kubernetes *readiness probes* lets Kubernetes tell whether a particular application is *ready*. For example, consider an application that, when it starts, has to perform a lengthy startup procedure. When the application is started by Kubernetes, it is not immediately ready. It cannot handle requests or workloads until the startup procedure is complete.

A readiness probe defines an action that Kubernetes takes in order to determine if the application running in a particular container is ready or not. There are several forms that a readiness probe can take, such as executing a command within the container, or issuing a `GET` request on a socket into a container. Whichever approach is defined, the end result is that the probe must give a `Yes` or `No` answer as to whether the application in the container is ready. The definition of ready is up to the application itself, and varies from application to application. In addition, the logic involved in determining if a given application is ready varies from application to application. See:

https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

In TimesTen Classic, the TimesTen Operator supports readiness probes in the following way: A TimesTen instance that is created by the Operator, and is running in the `tt` container of a Pod, is ready if the database inside of that Pod is loaded, replication is configured, and the database is either a fully functional active or a fully functional standby. A database that is down, or a database that is in the process of being created or duplicated is not ready.

A good use for readiness probes is if you want to replace one or more Nodes in your Kubernetes cluster. In such a case, you can cause Kubernetes to *drain* the workload from the Node. This causes Kubernetes to delete any Pods that are running on that Node, and create new Pods on other Nodes in the cluster to replace them.

Kubernetes supports the concept of a *Pod disruption budget*, where you can specify a budget for your application. This budget tells Kubernetes how many evicted Pods in a given Deployment can be tolerated. For example, assume you configure a Deployment with 20 replicas of your application. You could tell Kubernetes to tolerate up to 5 of the replicas being down at a time. When moving a workload from one Node to another, Kubernetes is careful not to delete more than 5 replicas at a time, and waits for their replacements to become ready before deleting more. See:

https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

https://kubernetes.io/docs/tasks/run-application/configure-pdb/

In the case of the TimesTen Kubernetes Operator, if you define a readiness probe in this way and you use a Pod disruption budget of `1` on TimesTen, you can drain the workload from one or more Nodes without creating a total TimesTen outage. Note that when Kubernetes deletes a Pod that is running TimesTen in TimesTen Classic, Kubernetes does not know if the Pod contains an active database or a standby database. Therefore, it may choose to delete the Pod that contains the active database. This causes a failover to the standby, which disrupts applications if performed during normal hours. There is not a way to prevent this. However, Kubernetes will not proceed to delete the other database until the one that was deleted comes back up and is completely in the `Healthy` state. See "Monitoring the health of each Pod in the active standby pair" for more information on the health of a Pod and the `Healthy` state.

TimesTen readiness probe

A TimesTen container is defined as `ready` if the TimesTen database it contains is either a fully functional active or a fully functional standby. If the database is down or is in a transitional state, the container is not `ready`.

A TimesTen container is ready to Kubernetes by the presence or absence of the `/tmp/readiness` file in the container's filesystem. If the file exists, the container is ready. You define the readiness probe in the TimesTenClassic object definition.

In the following example, the `sample` TimesTenClassic object includes a TimesTen readiness probe. Kubernetes executes the `cat` command in the `tt` container every `10` seconds. If the command exits with a return code of `0`, the container is ready. If the command returns any other value, the container is not ready.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
…
  template:
    spec:
…
      containers:
      - name: tt
        readinessProbe:
          exec:
            command:
            - cat
            - /tmp/readiness
          failureThreshold: 1
          periodSeconds: 10
          successThreshold: 1
```

4 Specify CPU and Memory Requests and Limits

This chapter discusses the importance of specifying CPU and memory requests and limits for all TimesTen Classic and TimesTenScaleout objects. It also provides an understanding of Linux cgroups and gives background information about the Linux `out of memory` (OOM) killer.

Topics:

About resource requests and limits

One of the core Linux technologies that is used to implement containers is cgroups. Cgroups can be used to enforce CPU and memory use limitations on a process or processes.

Kubernetes provides facilities that let you specify the amount of CPU and memory that a container consumes. When specified, Kubernetes uses your requests to do the following:

- Determine the node a particular Pod should be created on: Kubernetes determines which nodes have enough free resources to satisfy your request.
- Enforce these limits at runtime: Kubernetes ensures that applications do not exceed their requests.

Kubernetes chooses which node of the cluster to run a Pod on based on the Pod's resource requests, and the resources available on each node in the cluster. But once a Pod is scheduled onto a node, Kubernetes does not directly enforce limits at runtime. Rather, Kubernetes passes requests to Linux through cgroups. The node's Linux kernel then enforces the limits on Kubernetes behalf.

Let's look at an example of a Pod with the following definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: samplePod
spec:
  containers:
  - image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
```

```
    name: sample
    resources:
      limits:
        cpu: 20
        memory: 41Gi
      requests:
        cpu: 20
        memory: 41Gi
```

Kubernetes determines which node it will run the Pod on. It does this by examining the Pod's resources requests to determine which node has enough free resources to accommodate the Pod.

Once the node is determined, Kubernetes creates a `cpu` cgroup for the container (`sample` in this example) and configures the cgroup to have a limit of 20 CPUs. Kubernetes also creates a `memory` cgroup for the container and configures it to have a limit of 200 gigabytes. Kubernetes then forks off the lead process of the newly created container, and associates the initial process with the `cpu` and `memory` cgroups.

Since the lead process is associated with or running under these cgroups, that process and all its children are subject to the limits that the cgroups define. The Linux kernel on the node where the container is running automatically enforces these limits without any involvement from Kubernetes.

CPU limits are easily enforced by Linux. If an application wants to use more CPU than its limit, the kernel can choose not to dispatch the application for a period of time to keep its usage under control.

Memory limit enforcement is different than CPU enforcement. Linux has a component called the `out of memory` (OOM) killer. If processes exceed their intended memory usage, or if the system gets stressed, the OOM killer terminates processes abruptly.

Next, let's consider a Pod with this definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: samplePod
spec:
  containers:
  - image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    name: sample
```

In this Pod, there are no specified memory limits. The container runs in default cgroups with infinite limits. This cgroup is shared by all processes on the node that have no specified memory limit, whether the processes are running in containers or not. In this case, if the Linux node becomes memory constrained, the OOM killer chooses processes from the entire node to terminate. The victim might be TimesTen, it might be the Kubernetes kubelet, it might be some other process, or it might be all of them.

About TimesTen containerized deployments

As an in-memory database, TimesTen uses a large amount of memory by design. Processes that run TimesTen may be the largest memory users on a given node. If the operating system gets stressed, TimesTen is likely a prime candidate to get terminated by the Linux OOM killer. Therefore, it is crucial that memory requests and limits be specified for your TimesTen containers. TimesTen also recommends that CPU requests and limits be specified.

Note:

Ensure to set requests and limits for a resource (such as `memory` or `cpu`) to the same value.

The `memory` request and `memory` limit for the `tt` container is the most essential and crucial to specify. The value is dependent on the memory required to hold the TimesTen database as well as the memory required for the TimesTen daemon, subdaemons, cache and replication agents, Client/Server server, and so on. The memory required to hold the database is dependent on the size of your database.

The TimesTen Operator provides functionality to accurately size your TimesTen database. This functionality is discussed later. The additional memory required for the TimesTen daemon, subdaemons, Client/Server server is dependent on your SQL and PL/SQL usage and the memory requirements vary with your workload. The Operator provides a default of `1Gi` for this additional memory. You can use this default or change it. How to change it is discussed later.

Let’s take a look at the defaults for the TimesTen containers. In all cases, you have the option of changing the default.

- `tt` container:
  - `memory`: This value is discussed in detail throughout this chapter.
  - `cpu`: The value is dependent on how much CPU the `tt` container requires. This includes CPU used by the TimesTen daemon, subdaemons, replication agents, cache agents, and by the Client/Server server. The Client/Server server executes SQL on behalf of your applications, so the value depends on the workload. There is no default.
- `daemonlog` container:
  - `memory`: The default is `20Mi`.
  - `cpu`: The default is `200m`.
- `exporter` container (if provisioned):
  - `memory`: The default is `200Mi`.
  - `cpu`: The default is `200m`.

In addition, if you are using TimesTen Scaleout, there are additional TimesTen containers to consider:

- `tt` container of the management instance:
  - `memory`: The default is `1Gi`.

  - `cpu`: The default is `1`.
- `zookeeper` container:
  - `memory`: The default is `1Gi`.
  - `cpu`: The default is `500m`.

About specifying requests and limits for TimesTen containers

Let's explore how the TimesTen Operator sets CPU and memory requests and limits for TimesTen containers.

By default, and whenever possible, the Operator sets CPU and memory requests and limits for you. TimesTen recommends that you take this approach. An alternative approach is for you to disable this behavior and set the requests and limits yourself.

There are specific datum in the `.spec.ttspec` fields of the TimesTenClassic and TimesTenScaleout object custom resource definitions for both approaches. For details about the datum, see the table entries in TimesTenClassicSpecSpec and TimesTenScaleoutSpecSpec.

Here is a summary list:

- `automaticMemoryRequests`
- `databaseCPURequest`
- `databaseMemorySize`
- `additionalMemoryRequest`
- `memoryWarningPercent`
- `daemonlogMemoryRequest`
- `daemonlogCPURequest`
- `exporterMemoryRequest`
- `exporterCPURequest`
- `mgmtMemoryRequest`: (TimesTen Scaleout)
- `mgmtCPURequest`: (TimesTenScaleout)
- `zookeeperMemoryRequest`: (TimesTen Scaleout)
- `zookeeperCPURequest`:)TimesTen Scaleout)

Next, let's look at the approaches for specifying `cpu` and `memory` requests and limits to Kubernetes.

Approach 1: Operator determines requests and limits

This approach lets the Operator determine the appropriate `cpu` and `memory` requests and limits for Kubernetes. TimesTen recommends this approach.

This is also the default behavior (`.spec.ttspec.automaticMemoryRequests` set to `true`) for a TimesTenClassic or TimesTenScaleout object.

The Operator uses the datum in the `.spec.ttspec` fields of the TimesTenClassic and TimesTenScaleout object custom resource definitions to determine the CPU and memory requests and limits for each of the TimesTen containers. The Operator then provides this information to Kubernetes.

While it is important to specify `cpu` and `memory` requests/limits for TimesTen containers, there are defaults provided if you do not do so. See About specifying requests and limits for TimesTen containers.

However, it is essential that the value for the `memory` request/limit for the `tt` container that holds the TimesTen database be accurate.

Recall that the `memory` request/limit for the `tt` container is based on:

- Shared memory for the database: This is dependent on the size of the database.
- Additional memory: This is the memory required in addition to the database. It includes memory that is used for the TimesTen daemon, subdaemons, agents, Client/Server server.

The Operator provides the `.spec.ttspec.databaseMemorySize` and `.spec.ttspec.additionalMemoryRequest` datum for these specific memory requirements. The `.spec.ttspec.databaseMemorySize` is used to specify the size of the database and the `.spec.ttspec.additionalMemoryRequest` is used for the additional memory.

The Operator adds the value of `.spec.ttspec.additionalMemoryRequest` to the value of `.spec.ttspec.databaseMemorySize`. The sum is the `memory` request and `memory` limit to Kubernetes.

You do not have to specifically specify the `.spec.ttspec.databaseMemorySize` datum for a TimesTenClassic or TimesTenScaleout object. If not specified, the Operator attempts to determine the appropriate value.

TimesTen provides the `ttShmSize` utility to determine the shared memory requirements of a database, given its `sys.odbc.ini` entry. For information about `ttShmSize`, see ttShmSize in the *Oracle TimesTen In-Memory Database Reference*.

The equivalent for `sys.odbc.ini` in the Operator is the `db.ini` metadata file. You can provide the `db.ini` file in several ways:

- Embed in a ConfigMap referenced in `.spec.ttspec.dbConfigMap`.
- Embed in a Secret referenced in `.spec.ttspec.dbSecret`.
- Use an init container.

For details about the facilities that you can use to provide metadata files, see Populating the /ttconfig directory.

TimesTen recommends that you provide the `db.ini` metadata file in either a ConfigMap or a Secret. The Operator examines the Configmaps and Secrets, if any, in your TimesTenClassic or TimesTenScaleout objects. If the `db.ini` is found in a Configmap or Secret, the Operator uses the TimesTen `ttShmSize` utility to determine the appropriate amount of shared memory to request based on your database definition. This value is then used for the `.spec.ttspec.databaseMemorySize` value. With this approach, the Operator does the database sizing for you.

Let's look at an example:

```
kind: ConfigMap
metadata:
  name: resource9
data:
  adminUser: |
    adminuser/adminuserpwd
  schema.sql: |
    create user sampleuser identified by sampleuserpwd;
    grant admin to sampleuser;
    create table sampleuser.a (b number not null primary key, c
number, d timestamp);
    insert into sampleuser.a values(-1, -1, sysdate);
  db.ini: |
    Permsize=32768
    TempSize=4096
    LogBufMB=1024
    Connections=2048
    DatabaseCharacterSet=AL32UTF8
---
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: recommendedoption
spec:
  ttspec:
    dbConfigMap:
    - option1
    storageClassName: standard
    storageSize: 200Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    prometheus:
      insecure: true
```

In this case, the Operator runs `ttShmSize` against your provided `db.ini` file and determines the value for `.spec.ttspec.databaseMemorySize` automatically. The Operator then adds this value to the value for `.spec.ttspec.additionalMemoryRequest`. The sum is the `memory` request and `memory` limit to Kubernetes.

If you provide the `db.ini` file by using an init container or other mechanism, the Operator cannot determine the value for `.spec.ttspec.databaseMemorySize`. By the time the Pod is provisioned and the init container is executed, the Pod has already been created and its memory requirements defined. In such cases, you must manually provide the `.spec.ttspec.databaseMemorySize` as part of your YAML.

TimesTen recommends that you use the `ttShmSize` utility in a TimesTen instance outside of Kubernetes to determine the appropriate value for `.spec.ttspec.databaseMemorySize`. You do not need to create the database.

This example assumes you have created a TimesTen instance outside of Kubernetes and have created a DSN in your `sys.odbc.ini file` with the name `database1`. Use the `ttShmSize` utility based on provided values for the `PermSize`, `TempSize`, `LogBufMB`, and `Connections` connection attributes.

```
ttShmSize -connstr
"DSN=database1;PermSize=32768;TempSize=4096;LogBufMB=1024;Connections=2048"
The required shared memory size is 39991547720 bytes.
```

Specify this value in the `.spec.ttspec.databaseMemorySize` datum.

Let's look at an example that uses an init container and uses the calculated value for `.spec.ttspec.databaseMemorySize`.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: init1
spec:
  ttspec:
    databaseMemorySize: 41Gi
    storageClassName: standard
    storageSize: 200Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    prometheus:
      insecure: true
  template:
    spec:
      initContainers:
      - name: init1a
         command:
        - sh
        - "-c"
        - |
          /bin/bash <<'EOF'
          echo adminuser/adminuserpwd > /ttconfig/adminUser
          echo PermSize=32768 > /ttconfig/db.ini
          echo TempSize=4096 > /ttconfig/db.ini
          echo LogBufMB=1024 > /ttconfig/db.ini
          echo Connections=2048 > /ttconfig/db.ini
          echo DatabaseCharacterSet=AL32UTF8 >> /ttconfig/db.ini
          ls -l /ttconfig
          EOF
        volumeMounts:
        - name: tt-config
          mountPath: /ttconfig
      volumes:
      - name: tt-config
        emptyDir: {}
```

In this case, the Operator uses the value you specified for `.spec.ttspec.databaseMemorySize` to determine the size of the shared memory segment to hold the TimesTen database. The Operator then adds this value to the value for `.spec.ttspec.additionalMemoryRequest`. The sum is the `memory` request and `memory` limit to Kubernetes.

Note:

In TimesTen Scaleout, the `sys.odbc.ini file` (and corresponding `db.ini` file) define the size of a single element of the database, not the entire database. When provisioning a TimesTenScaleout object, the Operator uses the provided data in the same manner as it does for a TimesTenClassic object.

Approach 2: Use templates for requests and limits

This approach uses a template for specifying resource requests and limits. To enable this behavior, set the `.spec.ttspec.automaticMemoryRequests` datum to `false` for your TimesTenClassic or TimesTenScaleout object.

In your YAML for TimesTenClassic and TimesTenScaleout objects, you can specify a `template` for Pods. In this `template`, you specify attributes of various containers in the Pods, including the `tt` container. If you specify a `template` for one or more containers, the resource requests and limits for the containers are used by Kubernetes.

TimesTen recommends that you use the `ttShmSize` utility in a TimesTen instance outside of Kubernetes to determine the appropriate value for the `memory` request and limit for the `tt` container. You do not need to create the database.

This example assumes you have created a TimesTen instance outside of Kubernetes and have created a DSN in your `sys.odbc.ini file` with the name `database1`. Use the `ttShmSize` utility based on provided values for the `PermSize`, `TempSize`, `LogBufMB`, and `Connections` connection attributes.

```
ttShmSize -connstr
"DSN=database1;PermSize=32768;TempSize=4096;LogBufMB=1024;Connections=2
048"
The required shared memory size is 39991547720 bytes.
```

For information about `ttShmSize`, see ttShmSize in the *Oracle TimesTen In-Memory Database Reference*.

Let's look at an example:

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: resource1
spec:
  ttspec:
    storageClassName: standard
    storageSize: 100Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    prometheus:
      insecure: true
  template:
    spec:
```

```
containers:
- name: tt
  resources:
    requests:
      memory: "41Gi"
      cpu:    "20"
    limits:
      memory: "41Gi"
      cpu:    "20"
- name: daemonlog
  resources:
    requests:
      memory: "30Mi"
      cpu:    "210"
    limits:
      memory: "30Mi"
      cpu:    "210"
- name: exporter
  resources:
    requests:
      memory: "22Mi"
      cpu:    "310m"
    limits:
      memory: "22Mi"
      cpu:    "310m"
```

In this example, `cpu` and `memory` requests and limits for the `tt`, `daemonlog`, and `exporter` containers are included in the template. These resources are specified to Kubernetes.

About specifying the requests and limits to Kubernetes

The TimesTen Operator follows a specific order in determining the `cpu` and `memory` request and limits to Kubernetes:

- For the `tt` container that holds your TimesTen database, the Operator looks for the following in this order:
  - If you specify a template, the Operator uses the values in it.
  - If you specify `.spec.ttspec.databaseMemorySize`, the Operator uses its value.
  - If there is a `db.ini` file, the Operator uses the values in it.
  - If none of the above are true, the Operator uses the default.

  In addition, if you specified a value for `.spec.ttspec.databaseCPURequest`, that value is used as the `cpu` request and `cpu` limit to Kubernetes.
- If you specify resource requests or limits for the `daemonlog` container in your container templates in your TimesTenClassic or TimesTenScaleout object, the Operator honors those requests. If you do not, the Operator uses the values you supply in your object's `.spec.ttspec.daemonLogMemoryRequest` and `.spec.ttspec.daemonLogCPURequest` datum.
- If you specify resource requests or limits for the `exporter` container in your container templates in your TimesTenClassic or TimesTenScaleout object, the Operator honors those requests. If you do not, the Operator uses the values you supply in your

object's `.spec.ttspec.exporterMemoryRequest` and `.spec.ttspec.exporterCPURequest` datum.

- If you specify a template for the `tt` container in the `mgmtTemplate` in a TimesTenScaleout object, the Operator uses the resource data in that template. If you do not, the Operator uses the values you supply in your object's `.spec.ttspec.mgmtMemoryRequest` and `.spec.ttspec.mgmtCPURequest` datum.
- If you specify a template for the `zookeeper` container in the `zookeeperTemplate` in a TimesTenScaleout object, the Operator uses the resource data in that template. If you do not, the Operator uses the values you supply in your object's `.spec.ttspec.zookeeperMemoryRequest` and `.spec.ttspec.zookeeperCPURequest` datum.

About verifying databaseMemorySize

Whether specified by you or determined by the Operator, before a database is created, the Operator and the TimesTen agent checks that the `tt` containers in the relevant TimesTen Pods have the `memory` resources required to create the database. The Operator accomplishes this by running the TimesTen `ttShmSize` utility and comparing it with the memory quotas in the `tt` container's cgroup.

If the required resources are not available, the Operator returns an error message (as an Event) and moves the TimesTenClassic or TimesTenScaleout object to the `Failed` state.

This checking is performed even if the value of `.spec.ttspec.automaticMemoryRequests` is `false`.

About runtime memory monitoring

The TimesTen Operator monitors the memory usage of TimesTen Pods at runtime. It informs you of the following:

- If any `tt` containers are approaching their specified memory limits.
- If any TimesTen containers have been terminated by the Linux OOM killer.

Kubernetes Events are generated to report on these conditions.

Every `.spec.ttspec.pollingInterval` seconds, the TimesTen agent in each `tt` container queries the container's underlying Linux cgroup to determine the cgroup's `memory.limit_in_bytes` and `memory.usage_in_bytes` and reports these values to the Operator. The Kubernetes status of each container is similarly queried. The Operator uses this data to generate appropriate Events as needed.

If the `usage` is greater than `.spec.ttspec.memoryWarningPercent` of the `limit`, an Event is generated to notify you. In addition, if the Operator observes that one or more of the TimesTen related containers have been terminated or restarted (by the OOM killer or otherwise), the Operator reports the observation by using appropriate Events.

5

Deploying TimesTen Classic Databases

This chapter discusses the process for deploying active standby pairs of TimesTen databases. It describes the process for creating TimesTenClassic objects in your environment. It also provides examples that demonstrate how to monitor the provisioning of the active standby pair of TimesTen databases. The chapter concludes with examples that show you how to connect to the database and run operations in it.

Topics:

- Understanding the deployment process
- Defining and creating the TimesTenClassic object
- Monitoring the progress of the active standby pair deployment

Understanding the deployment process

The TimesTen Operator extends the Kubernetes API to provide the TimesTenClassic object type. This type provides the definitions you need to successfully deploy your TimesTen databases to the Kubernetes cluster. You customize these definitions for your particular environment. Specifically, you create a YAML file and, in it, you specify the required TimesTenClassic definitions for the TimesTenClassic object. By assigning values to the fields of these definitions, you customize and define your deployment environment. For example, when you supply the `oci` value for the `storageClassName` field, you are telling the Operator the name of the storage class you want to use. See "The TimesTen Operator Object Types" for the object definitions, and the fields that you define in your YAML file.

Examples of the YAML file were introduced previously when discussing ConfigMaps and Secrets, an init container, and other configuration options. (See "Using ConfigMaps and Secrets" for information on ConfigMaps and Secrets. Also see "Using an init container" and "Additional configuration options" for other configuration options.) However, "Defining and creating the TimesTenClassic object" shows you how to define the TimesTenClassic object in detail.

After specifying your configuration in the YAML file, you use the `kubectl create` command from your Linux development host to create the corresponding TimesTenClassic object in your cluster. After you issue this command, the process for deploying your active standby pair of TimesTen databases begins. You can view this process by issuing `kubectl get` and `kubectl describe` commands, such as, `kubectl get pods` and `kubectl describe timestenclassic`. Once your databases are deployed, you can then connect to your active database, issue queries, and perform other operations to verify your database is working as it should.

Defining and creating the TimesTenClassic object

Defining your environment involves creating TimesTenClassic objects with attributes customized for your environment. The fields include the name of the image pull secret, the name of your TimesTen image, and the other definitions required to successfully deploy your

TimesTen databases. See "About the TimesTenClassic object type" for information on defining objects of type TimesTenClassic.

Perform these steps to define and create the TimesTenClassic object:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (In this example, `sample`.) The YAML file contains the definitions for the TimesTenClassic object. See "TimesTenClassicSpecSpec" for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   In this example, replace the following. (The values you can replace are represented in **bold**.)

   - `name`: Replace `sample` with the name of your TimesTenClassic object.
   - `storageClassName`: Replace `oci` with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
   - `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. (This example assumes a production environment and uses 250Gi for storage. For demonstration purposes, you can use 50Gi of storage. See the `storageSize` and the `logStorageSize` entries in "Table 15-3" for information.)
   - `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location and name of your image. In this example, `container-registry.oracle.com/timesten` is the location of the image registry and `timesten:22.1.1.9.0` is the name of the image.
   - `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.
   - `dbConfigMap`: This example uses one ConfigMap (called `sample`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files. This ConfigMap will be included in the ProjectedVolume. This volume is mounted as `/ttconfig` in the TimesTen containers. See "Using ConfigMaps and Secrets" and "Example using one ConfigMap" for information on ConfigMaps.

   ```
   % vi sample.yaml

   apiVersion: timesten.oracle.com/v1
   kind: TimesTenClassic
   metadata:
     name: sample
   spec:
     ttspec:
       storageClassName: oci
       storageSize: 250Gi
       image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
       imagePullSecret: sekret
       dbConfigMap:
       - sample
   ```

2. Use the `kubectl create` command to create the TimesTenClassic object from the contents of the YAML file (in this example, `sample.yaml`). Doing so begins the process of deploying your active standby pair of TimesTen databases in the Kubernetes cluster.

   ```
   % kubectl create -f sample.yaml
   timestenclassic.timesten.oracle.com/sample created
   ```

You successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your TimesTen databases begins, but is not yet complete.

Monitoring the progress of the active standby pair deployment

You can use various `kubectl` commands to monitor the progress of the active standby pair deployment. After the deployment is complete and successful, you can connect to the database and run operations in it to verify it is working as it should.

- Monitor the state of TimesTenClassic
- Verify the underlying objects exist
- Verify connection to the active database

Monitor the state of TimesTenClassic

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the active standby pair as it is provisioned.

Note:

For the `kubectl get timestenclassic` and `kubectl describe timestenclassic` commands, you can alternatively specify `kubectl get ttc` and `kubectl describe ttc` respectively. `timestenclassic` and `ttc` are synonymous when used in these commands, and return the same results. The first `kubectl get` and the first `kubectl describe` examples in this chapter use `timestenclassic`. The remaining examples in this book use `ttc` for simplicity.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

```
% kubectl get timestenclassic sample
NAME     STATE          ACTIVE   AGE
sample   Initializing   None     11s
```

2. Use the `kubectl describe` command to view the initial provisioning in detail.

```
% kubectl describe timestenclassic sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2022-04-30T15:35:12Z
  Generation:          1
  Resource Version:    20231755
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
  UID:                 517a8646-a354-11ea-a9fb-0a580aed5e4a
Spec:
  Ttspec:
    Db Config Map:
      sample
    Image:                container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250Gi
Status:
  Active Pods:       None
  High Level State:  Initializing
  Last Event:        3
  Pod Status:
    Cache Status:
      Cache Agent:        Down
      Cache UID Pwd Set:  false
      N Cache Groups:     0
    Db Status:
      Db:            Unknown
      Db Id:         0
      Db Updatable:  Unknown
    Initialized:     true
    Pod Status:
      Agent:                  Down
      Last Time Reachable:  0
      Pod IP:
      Pod Phase:              Pending
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Down
      Rep Peer P State:             Unknown
      Rep Scheme:                   Unknown
      Rep State:                    Unknown
    Times Ten Status:
      Daemon:          Down
      Instance:        Unknown
      Release:         Unknown
    Admin User File:   false
    Cache User File:   false
    Cg File:           false
    High Level State:  Down
    Intended State:    Active
    Name:              sample-0
    Schema File:       false
    Cache Status:
      Cache Agent:        Down
      Cache UID Pwd Set:  false
      N Cache Groups:     0
    Db Status:
      Db:            Unknown
      Db Id:         0
      Db Updatable:  Unknown
    Initialized:     true
    Pod Status:
      Agent:                  Down
      Last Time Reachable:  0
      Pod IP:
      Pod Phase:              Pending
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Down
      Rep Peer P State:             Unknown
      Rep Scheme:                    Unknown
      Rep State:                     Unknown
    Times Ten Status:
      Daemon:             Down
      Instance:           Unknown
      Release:            Unknown
    Admin User File:      false
    Cache User File:      false
    Cg File:              false
    High Level State:     Unknown
    Intended State:       Standby
    Name:                 sample-1
    Schema File:          false
  Rep Create Statement:   create active standby pair "sample" on
 "sample-0.sample.mynamespace.svc.cluster.local", "sample" on
 "sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store "sample" on
 "sample-0.sample.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
 store "sample" on "sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD  0
  Rep Port:               4444
  Status Version:         1.0
Events:
  Type  Reason  Age   From        Message
  ----  ------  ----  ----        -------
  -     Create  50s   ttclassic   Secret tt517a8646-a354-11ea-a9fb-0a580aed5e4a
created
  -     Create  50s   ttclassic   Service sample created
  -     Create  50s   ttclassic   StatefulSet sample created
```

3. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

```
% kubectl get ttc sample
NAME      STATE     ACTIVE     AGE
sample    Normal    sample-0   3m5s
```

4. Use the `kubectl describe` command again to view the active standby pair provisioning in detail.

   Note: In this example, the `now Normal` line displays on its own line. In the actual output, this line does not display as its own line, but at the end of the `StateChange` previous line.

```
% kubectl describe ttc sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2022-04-30T15:35:12Z
  Generation:          1
  Resource Version:    20232668
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
  UID:                 517a8646-a354-11ea-a9fb-0a580aed5e4a
Spec:
  Ttspec:
    Db Config Map:
      sample
    Image:             container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250Gi
Status:
  Active Pods:       sample-0
  High Level State:  Normal
  Last Event:        35
  Pod Status:
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:            Loaded
      Db Id:         26
      Db Updatable:  Yes
    Initialized:     true
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1590939597
      Pod IP:               192.0.2.1
      Pod Phase:            Running
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    ACTIVE
    Times Ten Status:
      Daemon:          Up
      Instance:        Exists
      Release:         22.1.1.9.0
    Admin User File:   true
    Cache User File:   false
    Cg File:           false
    High Level State:  Healthy
    Intended State:    Active
    Name:              sample-0
    Schema File:       true
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:            Loaded
      Db Id:         26
      Db Updatable:  No
    Initialized:     true
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1590939597
      Pod IP:               192.0.2.2
      Pod Phase:            Running
    Replication Status:
      Last Time Rep State Changed:  1590939496
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Times Ten Status:
      Daemon:           Up
      Instance:         Exists
      Release:          22.1.1.9.0
    Admin User File:    true
    Cache User File:    false
    Cg File:            false
    High Level State:   Healthy
    Intended State:     Standby
    Name:               sample-1
    Schema File:        true
  Rep Create Statement:  create active standby pair "sample" on
"sample-0.sample.mynamespace.svc.cluster.local", "sample" on
"sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store "sample" on
"sample-0.sample.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
store "sample" on "sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD 0
  Rep Port:             4444
  Status Version:       1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     Create       4m43s  ttclassic  Secret tt517a8646-a354-11ea-
a9fb-0a580aed5e4a created
  -     Create       4m43s  ttclassic  Service sample created
  -     Create       4m43s  ttclassic  StatefulSet sample created
  -     StateChange  3m47s  ttclassic  Pod sample-0 Daemon Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-0 CacheAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-0 RepAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 Daemon Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 CacheAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 RepAgent Unknown
  -     StateChange  3m26s  ttclassic  Pod sample-0 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-0 Release 22.1.1.9.0
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-1 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-1 Release 22.1.1.9.0
  -     StateChange  3m26s  ttclassic  Pod sample-1 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Up
  -     StateChange  3m25s  ttclassic  Pod sample-1 Daemon Up
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  2m13s  ttclassic  Pod sample-0 Database Updatable
  -     StateChange  2m13s  ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepAgent Not Running
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepScheme None
  -     StateChange  2m13s  ttclassic  Pod sample-0 Database Loaded
  -     StateChange  2m11s  ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  2m10s  ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  2m10s  ttclassic  Pod sample-0 RepState ACTIVE
  -     StateChange  113s   ttclassic  Pod sample-1 Database Loaded
  -     StateChange  113s   ttclassic  Pod sample-1 Database Not Updatable
  -     StateChange  113s   ttclassic  Pod sample-1 CacheAgent Not Running
  -     StateChange  113s   ttclassic  Pod sample-1 RepAgent Not Running
  -     StateChange  113s   ttclassic  Pod sample-1 RepScheme Exists
  -     StateChange  113s   ttclassic  Pod sample-1 RepState IDLE
  -     StateChange  106s   ttclassic  Pod sample-1 RepAgent Running
  -     StateChange  101s   ttclassic  Pod sample-1 RepState STANDBY
  -     StateChange  101s   ttclassic  TimesTenClassic was Initializing, now Normal
```

Your active standby pair of TimesTen databases are successfully deployed (as indicated by `Normal`.) There are two TimesTen databases, configured as an active standby pair. One database is active. (In this example, `sample-0` is the active database, as indicated by `Rep`

State ACTIVE). The other database is standby. (In this example, `sample-1` is the standby database as indicated by `Rep State STANDBY`). The active database can be modified and queried. Changes made on the active database are replicated to the standby database. If the active database fails, the Operator automatically promotes the standby database to be the active. The formerly active database will be repaired or replaced, and will then become the standby.

Verify the underlying objects exist

The Operator creates other underlying objects automatically. Verify that these objects are created.

1. StatefulSet:

```
% kubectl get statefulset sample
NAME      READY   AGE
sample    2/2     8m21s
```

2. Service:

```
% kubectl get service sample
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
sample    ClusterIP   None         <none>        6625/TCP   9m28s
```

3. Pods:

```
% kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
sample-0                                      2/2     Running   0          10m
sample-1                                      2/2     Running   0          10m
timesten-operator-5d7dcc7948-8mnz4            1/1     Running   0          11h
```

4. PersistentVolumeClaims (PVCs):

```
% kubectl get pvc
NAME                         STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
tt-persistent-sample-0       Bound
ocid1.volume.oc1.phx.abyhqljrbxcgzyixa4pmmcwiqxgqclc7gxvdnoty367w2qn26tij6kfp
x
6qq
250Gi      RWO            oci            10m
tt-persistent-sample-1       Bound
ocid1.volume.oc1.phx.abyhqljtt4qxxoj5jqiskriskh66hakaw326rbza4uigmuaezdnu53qh
h
oaa
250Gi      RWO            oci            10m
```

Verify connection to the active database

You can run the `kubectl exec` command to invoke shells in your Pods and control TimesTen, which is running in those Pods. By default, TimesTen runs in the Pods as the `timesten` user. Once you have established a shell in the Pod, verify you can connect to the `sample` database, and that the information from the metadata files is correct. You can optionally run queries against the database or any other operations.

1. Establish a shell in the Pod.

```
% kubectl exec -it sample-0 -c tt -- /bin/bash
```

2. Connect to the `sample` database. Verify the information in the metadata files is in the database correctly. For example, attempt to connect to the database as the `sampleuser` user. Check that the `PermSize` value of `200` is correct. Check that the `sampleuser.emp` table exists.

```
 % ttIsql sample

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)

Command> connect adding "uid=sampleuser;pwd=samplepw" as sampleuser;
Connection successful:
DSN=sample;UID=sampleuser;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
sampleuser: Command> tables;
  SAMPLEUSER.EMP
1 table found.
```

6

Deploying your TimesTen Grid

The TimesTen Operator supports the deployment of a TimesTen Scaleout grid and associated databases in your Kubernetes cluster.

This chapter provides background information about the deployment process. It summarizes the planning process for configuring a grid and shows you how to apply that plan when configuring a grid in the Kubernetes environment. It describes the steps the TimesTen Operator takes to deploy and provision a grid based on the information you provide. Use this information to gain an understanding of how the Operator functions when deploying your TimesTen Scaleout grid.

The second part of the chapter provides you with an end-to-end example that shows you the steps to deploy your grid in the Kubernetes cluster.

If you want to advance directly to the example, see Deploy your grid.

The TimesTen Operator also supports TimesTen Cache in TimesTen Scaleout. See Working with TimesTen Cache.

Topics:

- About deploying your grid
- Deploy your grid

About deploying your grid

The information in this section provides background information about configuring and deploying a grid in the Kubernetes environment.

Topics:

- About planning your grid
- About configuring your grid
- About provisioning your grid
- About ssh
- About creating your grid

About planning your grid

One of the features of the TimesTen Operator is the ability for it to provision a TimesTen Scaleout grid and its database in the Kubernetes cluster. Just like in any other TimesTen Scaleout environment, you must do some planning for your grid.

Here are some considerations:

- K-Safety (represented by `k`): How many copies of your TimesTen database? The K-Safety factor determines the number of data spaces in your grid. For example, if you set k to 2, there are two copies of your database: one copy resides in data space one and the second copy in data space two.

- Replica sets: How many replica sets in your grid? A replica set contains k elements, where each element in the replica set is an exact copy of the other elements in the replica set. The value of k, combined with the number of replica sets, determines the number of data instances in the grid. For example, if you set k to 2, and replica sets to 3, then there are six data instances in the grid.
- ZooKeeper instances: How many ZooKeeper instances for the grid?
- Database definition file (DbDef): What data store and first connection attributes are needed for the database in the grid?
- Direct connectable: What general connection attributes are needed for the database when using direct access?
- Client/server connectable: What general connection attributes are needed for the database when using client/server access?

After you define your configuration, you provide that information to the TimesTen Operator. The Operator takes over from there. It automatically does the provisioning and the deployment of the grid and database for you.

About configuring your grid

The TimesTen Operator provides the TimesTenScaleout object type and configuration metadata so that you can define and then deploy your TimesTen Scaleout grid and database. The TimesTenScaleout object type provides the syntax for configuring your grid. The configuration metadata lets you define the connection attributes for your database. Taken together, the Operator has the necessary information to provision your grid and database in the Kubernetes cluster.

Based on your planned configuration in About planning your grid, you can apply that configuration to the Kubernetes environment:

- K-Safety (k): The Operator provides the .spec.ttspec.k element as part of the syntax for the TimesTenScaleout object type. You specify k in the YAML manifest file for your TimesTenScaleout object.
- Replica sets: The Operator provides the .spec.ttspec.nReplicaSets element as part of the syntax for the TimesTenScaleout object type. You specify nReplicaSets in the YAML manifest file for your TimesTenScaleout object.
- ZooKeeper instances: The Operator provides the .spec.ttspec.nZookeeper element as part of the syntax for the TimesTenScaleout object type. You specify nZookeeper in the YAML manifest file for your TimesTenScaleout object.
- Database definition file (DbDef): The Operator creates a DbDef from the contents of the db.ini metadata file. You create this metadata file and then use a Kubernetes facility (or some other means) to place this file in the /ttconfig directory of the tt containers.
- One or more direct connectables: The Operator creates one or more direct connectables from the contents of the *.connect metadata files. You create one or more of these *.connect files and then use a Kubernetes facility (or some other means) to place the files in the /ttconfig directory of the tt containers.
- One or more client/server connectables: The Operator creates one or more client/server connectables from the contents of the *.csconnect metadata files. You

create one or more of these `*.csconnect` files and then use a Kubernetes facility (or some other means) to place the files in the `/ttconfig` directory of the `tt` containers.

See TimesTenScaleoutSpecSpec, List of the configuration metadata, and Populating the /ttconfig directory.

After the metadata files are placed in the `/ttconfig` directory of the `tt` containers, and you configure and then deploy your TimesTenScaleout object in the Kubernetes cluster, Kubernetes informs the Operator that a TimesTenScaleout object has been created. The Operator begins the process of creating additional Kubernetes objects in order to implement your grid.

About provisioning your grid

The TimesTen Operator gathers the information from the TimesTenScaleout object and begins instantiating the TimesTen Scaleout grid and database. It creates the following StatefulSets:

- One StatefulSet that provides the management instance for the grid. The Operator supports one management instance. The underlying Pod for this management instance is also created.
- One StatefulSet that provides one or more ZooKeeper instances. The Operator determines the number of ZooKeeper instances by the value you specified for the `nZookeeper` field in your TimesTenScaleout object definition. For example, if you specified a value of `3` for `nZookeeper`, the Operator creates one StatefulSet with three replicas. The underlying Pods for these ZooKeeper instances are also created. There is one Pod per ZooKeeper instance.
- Additional StatefulSets, the number of which is determined by the value you specified for the `k` field in your TimesTenScaleout object definition. For example, if you specified a value of `2` for `k`, the Operator creates two StatefulSets. These StatefulSets provide data instances for the grid. Each of the `k` StatefulSets provides Pods to implement a single data space in the resultant grid. The StatefulSet has `M` replicas, the number of which is determined by the value you specified for the `nReplicaSets` field in your TimesTenScaleout object definition. For example, if you set `nReplicaSets` to `3`, each StatefulSet has three replicas. This number of replicas determines the number of replica sets in the resultant grid.

  In the preceding example, one StatefulSet has three replicas. This one StatefulSet contains three data instances in data space one. A Pod is created for each data instance, so there are three Pods created. The second StatefulSet contains three data instances in data space two. A Pod is created for each data instance, so there are three Pods created. There are a total of six total Pods created for the six data instances.

In addition, the Operator creates the following Kubernetes headless Services:

- One headless Service that provides DNS names for the Pods that contain the management and data instances. This service enables client/server access to the Pods using the TimesTen client/server port `6625`.
- One headless Service that provides DNS names for the Pods that contain the ZooKeeper instances. This service enables access to the ZooKeeper internal ports `2888` and `3888`, as well as the external port `2181`.

There is an example of these StatefulSets and headless Services in Verify the underlying objects.

The Operator also creates Persistent Volume Claims (PVCs) for the Pods. These PVCs cause persistent volumes to be allocated by Kubernetes and attached to the TimesTen Pods and ZooKeeper Pods. See Persistent storage.

Pods that run ZooKeeper consists of a single container called `zookeeper`. Each Pod that is running TimesTen consists of at least two containers. The `tt` container runs the TimesTen agent and TimesTen. The `daemonlog` container writes the TimesTen daemon log to stdout.

As the `tt` containers in the TimesTen Pods start, the Operator assembles them into a working grid. The grid's model is configured with several objects, including:

- Hosts for the data instances in each of the data space groups
- Each host configured with an installation of TimesTen
- Each host configured with a single TimesTen instance
- A `DbDef` (the contents of which are from the `db.ini` file)
- Direct mode connectables, if any, (the contents of which are from the `*.connect` files)
- Client/server connectables, if any, (the contents of which are from the `*.csconnect` files)

About ssh

A TimesTen Scaleout grid relies on password-less ssh among the instances of the grid. The TimesTen Operator automatically configures password-less ssh among the `tt` containers in the grid in your Kubernetes environment. There is no intervention that you need to do.

About creating your grid

When creating the grid, the TimeTen Operator transitions to and from various High Level states. Here is an explanation of these states:

After you create your TimesTenScaleout object in the Kubernetes cluster, the TimesTen Operator creates the StatefulSets and Services that are required to deploy your TimesTenScaleout grid and database. The Operator assigns a High Level state of `Initializing` to the TimesTenScaleout object.

The Operator periodically polls the status of the StatefulSets' objects and their underlying Pods. When the ZooKeeper Pods are ready, the TimesTenScaleout object transitions from the `Initializing` state to the `ZooKeeperReady` state.

When the TimesTen agent in the management Pod starts up, the Operator instructs the agent to create a TimesTen instance and grid. The TimesTenScaleout object transitions to the `GridCreated` state.

The Operator waits for all of the TimesTen agents in all of the Pods to start. Once all have started, the Operator instructs the TimesTen agent in the management instance to create the hosts, the installations, and the instances in the grid's model for the data instances in the grid.

The `DbDef` is then created from the contents of the `db.ini` file. The direct connectables are created from the contents of the `*.connect` files. The client/server connectables are created from the contents of the `*.csconnect` files.

The model is applied and the data instances of the grid are created. The TimesTenScaleout object transitions to the `InstanceCreated` state.

The Operator then instructs the management instance to create the database (by using the TimesTen `ttGridAdmin` utility with the `dbCreate` option) and to create the initial distribution map (by using the TimesTen `ttGridAdmin` utility with the `dbDistribute -add all -apply` options). The TimesTenScaleout object then transitions to the `DbCreated` state.

The Operator then instructs the TimesTen agent in one data instance to run the TimesTen `ttIsql` utility to create the user in the `adminUser` file and run the `schema.sql` file (if you provided these files). The TimesTenScaleout object transitions to the `Normal` state.

The grid and databases are created. The TimesTen agent then opens the database for connections.

The Operator manages and monitors the TimesTenScaleout objects after they are deployed in your Kubernetes cluster. The Operator also detects, repairs, and recovers from failures in your grid and associated databases. See Managing TimesTen Scaleout.

There is an example showing the Operator transitioning to and from these High Level states in Monitor the High Level state of the TimesTenScaleout object.

Deploy your grid

This section provides a step by step walk-through that shows you how to create and deploy a TimesTen Scaleout grid and database in your Kubernetes cluster. The walk-through starts with an example that shows you how to create metadata files and create a Kubernetes ConfigMap. It continues with an example that shows you how to create a YAML file that contains the definitions for your TimesTenScaleout object. It shows you how to deploy that YAML file in your Kubernetes cluster. You learn how to monitor the provisioning of the TimesTen grid and database, and verify the underlying Kubernetes objects were created by the TimesTen Operator. The walk-through completes with one example that shows you how to connect to the TimesTen database and run operations in it. The final example shows you how to connect to the management instance to verify the health of the database and its elements.

- Create the configuration metadata and the Kubernetes ConfigMap for the grid
- Define and deploy the TimesTenScaleout object
- Monitor the High Level state of the TimesTenScaleout object
- Verify the underlying objects
- Connect to the database
- Manage the database

Create the configuration metadata and the Kubernetes ConfigMap for the grid

The following example creates the `db.ini`, the `adminUser`, and the `schema.sql` metadata files for the TimesTen grid and the database. The example also creates a direct and a client/server connectable. The example creates a Kubernetes ConfigMap to place these metadata files into the `/ttConfig` directory of the TimesTen containers.

Note:

You can use any Kubernetes mechanism to place these metadata files into the `/ttConfig` directory of the TimesTen containers. See Populating the /ttconfig directory.

On your development host, complete the following steps:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_scaleout` subdirectory. (The `cm_scaleout` directory is used in the remainder of this example to denote this directory.)

   ```
   mkdir -p cm_scaleout
   ```

2. Change to this ConfigMap directory.

   ```
   cd cm_scaleout
   ```

3. Create the `db.ini` file in this ConfigMap directory. In this example, define the `PermSize` and the `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ```

4. Create the `adminUser` file.

   ```
   vi adminUser

   adminuser/adminuserpwd
   ```

5. Create the `schema.sql` file. In this `schema.sql` file, create the `sampleuser` user, create the `s` sequence for the `sampleuser` user, and the `emp` table for the `sampleuser` user. The Operator automatically initializes each element of the TimesTen database with these object definitions.

   ```
   vi schema.sql

   create user sampleuser identified by sampleuserpwd;
   grant admin to sampleuser;
   create sequence sampleuser.s;
   create table sampleuser.emp (id number not null primary key, name
   char (32));
   ```

6. Create the `sampledirect` direct connectable.

   ```
   vi sampledirect.connect

   ConnectionCharacterSet=AL32UTF8
   ```

7. Create the `sampleclient` client/server connectable.

   ```
   vi sampleclient.csconnect
   
   TTC_Timeout=70
   ```

8. Optional: Verify the metadata files are present in the `cm_scaleout` ConfigMap directory.

   ```
   ls
   ```

   The output is the following:

   ```
   adminUser
   db.ini
   sampleclient.csconnect
   sampledirect.connect
   schema.sql
   ```

9. From the `cm_scaleout` directory, create the `samplescaleout` ConfigMap. The files in the `cm_scaleout` directory are included in the ConfigMap. These files are later available in the TimesTen containers. Replace `samplescaleout` with a name of your choosing.

   ```
   kubectl create configmap samplescaleout --from-file .
   ```

   The output is the following:

   ```
   configmap/samplescaleout created
   ```

10. Use the `kubectl describe` command to verify the contents of the `samplescaleout` ConfigMap.

   ```
   kubectl describe configmap samplescaleout
   ```

   The output is the following:

   ```
   Name:         samplescaleout
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   
   Data
   ====
   adminUser:
   ----
   adminuser/adminuserpwd
   
   db.ini:
   ----
   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   
   sampleclient.csconnect:
   ```

```
----
TTC_Timeout=70

sampledirect.connect:
----
ConnectionCharacterSet=AL32UTF8

schema.sql:
----
create user sampleuser identified by sampleuserpwd;
grant admin to sampleuser;
create sequence sampleuser.s;
create table sampleuser.emp (id number not null primary key, name
char (32));

Events:  <none>
```

You successfully created the metadata files and the ConfigMap.

Define and deploy the TimesTenScaleout object

Defining your environment involves creating your TimesTenScaleout object with attributes customized for your environment. These attributes are described in the steps below. For additional information on defining objects of type TimesTenScaleout, see TimesTen Scaleout.

To define and create the TimesTenScaleout object, complete the following steps:

1. Create a YAML file. You can choose any name for this YAML file, but you may want to use the same name you used for the name of the TimesTenScaleout object. (This example uses `samplescaleout`.) The YAML file contains the definitions for the TimesTenScaleout object. In this example, the fields that are specific to a TimesTenScaleout object are as follows:

   - `k`: Set the value of `k` to the number of copies of data for your TimesTen database. This value determines the number of StatefulSets that the TimesTen Operator creates. A StatefulSet provides the Pods that are used to implement a single data space in the grid. For example, if you set `k` to `2`, the Operator creates two StatefulSets. One StatefulSet provides the Pods for the data instances in data space one. The second StatefulSet provides the Pods for the data instances in data space two.

     This example sets `k` to `2`.

     For information on K-safety and determining an appropriate value for `k`, see K-safety in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

   - `nReplicaSets`: Set the value to the number of replica sets in the grid. A replica set contains `k` elements (where each element is an exact copy of the other elements in the replica set). The `nReplicaSets` value determines the number of replicas for each StatefulSet. For example, if you set `k` to `2`, the TimesTen Operator creates two StatefulSets for the data instances. If you set `nReplicaSets` to `3`, each StatefulSet contains three replicas, and the total number of replica sets in the database is three.

     This example sets `nReplicaSets` to `3`.

For information on replica sets, see Understanding replica sets in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

- `nZookeeper`: Set the value to the number of ZooKeeper Pods to provision in a StatefulSet. Your options are `1` or `3` or `5`.

  This example sets `nZookeeper` to `3`.

You also need to specify the following fields:

- `name`: Replace `samplescaleout` with the name of your TimesTenScaleout object.
- `storageClassName`: Replace `oci` with the name of the storage class in your Kubernetes cluster that is used to allocate Persistent Volumes to hold the TimesTen database.
- `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. (This example assumes a production environment and uses 250Gi for storage. For demonstration purposes, you can use 50Gi of storage.) See the `storageSize` and the `logStorageSize` entries in Table 15-7.
- `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location of the image registry and the name of the image. If you are using the Oracle `container-registry.oracle.com/timesten` registry as the image registry and the `timesten:22.1.1.9.0` image as the container image, no replacement is necessary.
- `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.
- `dbConfigMap`: This example uses one ConfigMap (called `samplescaleout`) for the `db.ini`, the `adminUser`, the `schema.sql`, the `sampledirect.connect`, and the `sampleclient.csconnect` metadata files. This ConfigMap is included in the Projected Volume. The volume is mounted as `/ttconfig` in the TimesTen containers. See Using ConfigMaps and Secrets.

```
vi samplescaleout.yaml

kind: TimesTenScaleout
metadata:
  name: samplescaleout
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - samplescaleout
    k: 2
    nReplicaSets: 3
    nZookeeper: 3
```

2. Create the TimesTenScaleout object from the contents of the YAML file. Doing so begins the process of creating and provisioning a TimesTen grid and database in your Kubernetes cluster.

```
kubectl create -f samplescaleout.yaml
```

The output is the following:

```
timestenscaleout.timesten.oracle.com/samplescaleout created
```

You successfully created the TimesTenScaleout object in the Kubernetes cluster. The process of provisioning your TimesTen grid and database begins, but is not yet complete.

Monitor the High Level state of the TimesTenScaleout object

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the grid and database creation.

> **Note:**
>
> For the `kubectl get timestenscaleout` command, you can alternatively specify `kubectl get tts`. When used in the `kubectl get` command, `timestenscaleout` and `tts` are synonymous, and return the same results. The first `kubectl get` examples in this chapter use `timestenscaleout`. For simplicity, the remaining examples in this book use `tts`.

1. Review the High Level state of the TimesTenScaleout object. Use the `kubectl get` command and observe the `STATE` field. Notice that the value is `Initializing`. The Operator has created the Kubernetes StatefulSets and Services. The process to deploy and provision your grid and database has begun, but is not yet complete. As you issue additional `kubectl get` commands, observe how the TimesTenScaleout object transitions to different states. For more information on these states, see About creating your grid.

   ```
   kubectl get tts samplescaleout
   ```

   The output is similar to the following:

   ```
   NAME            OVERALL        MGMT   CREATE   LOAD   OPEN   AGE
   samplescaleout  Initializing                                 20s
   ```

   ```
   kubectl get tts samplescaleout
   ```

   The output is similar to the following:

   ```
   NAME            OVERALL         MGMT   CREATE   LOAD   OPEN   AGE
   samplescaleout  ZookeeperReady
   2m48s
   ```

   ```
   kubectl get tts samplescaleout
   ```

The output is similar to the following:

```
NAME             OVERALL      MGMT   CREATE   LOAD   OPEN   AGE
samplescaleout   GridCreated                                3m58s
```

```
kubectl get tts samplescaleout
```

The output is similar to the following:

```
NAME             OVERALL           MGMT   CREATE   LOAD   OPEN   AGE
samplescaleout   InstancesCreated                                6m4s
```

```
kubectl get tts samplescaleout
```

The output is similar to the following:

```
NAME             OVERALL    MGMT     CREATE    LOAD               OPEN   AGE
samplescaleout   DatabaseCreated
6m10s
```

2. Use the `kubectl get` command again to observe if the High Level state has transitioned from `DatabaseCreated` to `Normal`. A `Normal` state indicates the grid and database are provisioned and the process is complete.

   ```
   kubectl get tts samplescaleout
   ```

   The output is similar to the following:

   ```
   NAME             OVERALL   MGMT     CREATE    LOAD               OPEN   AGE
   samplescaleout   Normal    Normal   created   loaded-complete    open
   6m52s
   ```

Your TimesTen Scaleout grid and database are successfully created and provisioned in your Kubernetes cluster. The database is open for connections.

Verify the underlying objects

The TimesTen Operator creates the following objects for the `samplescaleout` TimesTenScaleout object.

1. StatefulSets:

   ```
   kubectl get statefulsets
   ```

   The output is the following:

   ```
   NAME                    READY   AGE
   samplescaleout-data-1   3/3     26m
   samplescaleout-data-2   3/3     26m
   ```

```
samplescaleout-mgmt     1/1     26m
samplescaleout-zk       3/3     26m
```

The Operator creates the `samplescaleout-data-1` and the `samplescaleout-data-2` StatefulSets. Two StatefulSets are created because the value of `k` is set to `2`. Each StatefulSet provides the Pods for a single data space. There are two data spaces in the grid. Each of the StatefulSets has three replicas (`nReplicaSets` is set to `3`). Therefore, the number of replica sets in the grid is three. There are three data instances in data space one and three data instances in data space two.

The `samplescaleout-mgmt` StatefulSet provides the Pod for the management instance. There is one management instance.

The `samplscaleout-zk` StatefulSet provides the Pods for the ZooKeeper instances. There are three ZooKeeper instances (`nZookeeper` is set to `3`).

2. Pods:

```
kubectl get pods
```

The output is the following:

```
NAME                                    READY   STATUS    RESTARTS
AGE
samplescaleout-data-1-0                 2/2     Running   0
27m
samplescaleout-data-1-1                 2/2     Running   0
27m
samplescaleout-data-1-2                 2/2     Running   0
27m
samplescaleout-data-2-0                 2/2     Running   0
27m
samplescaleout-data-2-1                 2/2     Running   0
27m
samplescaleout-data-2-2                 2/2     Running   0
27m
samplescaleout-mgmt-0                   2/2     Running   0
27m
samplescaleout-zk-0                     1/1     Running   0
27m
samplescaleout-zk-1                     1/1     Running   0
26m
samplescaleout-zk-2                     1/1     Running   0
25m
timesten-operator-7677964df9-sp2zp      1/1     Running   0
3d20h
```

The `samplescaleout-data-1` StatefulSet contains the data instances in data space one. This StatefulSet creates the `samplescaleout-data-1-0`, `samplescaleout-data-1-1`, and `samplescaleout-data-1-2` Pods. The `samplescaleout-data-2` contains the data instances in data space two. This StatefulSet creates the `samplescaleout-data-2-0`, `samplescaleout-data-2-1`, and `samplescaleout-data-2-2` Pods.

The `samplescaleout-mgmt-0` StatefulSet contains the management instance for the grid. This StatefulSet creates the `samplescaleout-mgmt-0` Pod.

Each of the Pods previously mentioned run TimesTen. Each Pod contains at least two containers. In this example, each Pod contains two containers. The `tt` container runs the TimesTen agent and TimesTen. The `daemonlog` container writes the TimesTen daemon log to `stdout`.

The `samplescaleout-zk` StatefulSet contains the three ZooKeepers instances. This StatefulSet creates the `samplescaleout-zk-1`, `samplescaleout-zk-2`, and `samplescaleout-zk-3` Pods. Each of these Pods contain a single container called `zookeeper`.

The TimesTen Operator is running in the `timesten-operator-554887b4c-48zwk` Pod.

3. Headless Services:

```
kubectl get services
```

The output is the following:

```
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)                          AGE
samplescaleout       ClusterIP   None            <none>
6625/TCP                         28m
samplescaleout-zk    ClusterIP   None            <none>        2181/
TCP,2888/TCP,3888/TCP      28m
```

The Operator creates the `samplescaleout` Headless Service. This Service provides the DNS names for the Pods that contain the TimesTen management and data instances. The Service allows client/server access to the Pods that use the client/server port (`6625`). The DNS names are in the format: `*podname*.samplescaleout.*namespace*.svc.cluster.local`.

The following example shows the DNS names for the `samplescaleout` Headless Service:

```
samplescaleout-mgmt-0.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-1-0.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-1-1.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-1-2.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-2-0.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-2-1.samplescaleout.mynamespace.svc.cluster.local
samplescaleout-data-2-2.samplescaleout.mynamespace.svc.cluster.local
```

The Operator creates a second Headless Service called `samplescaleout-zk`. This Service allows access to the ZooKeeper internal ports (`2888` and `3888`) as well as the external port (`2181`). The DNS names are in the format: `samplescaleout-zk-*n*.samplescaleout-zk.*namespace*.svc.cluster.local`.

The following example shows the DNS names for the `samplescaleout-zk` Headless Service:

```
samplescaleout-zk-0.samplescaleout-zk.mynamespace.svc.cluster.local
samplescaleout-zk-1.samplescaleout-zk.mynamespace.svc.cluster.local
samplescaleout-zk-2.samplescaleout-zk.mynamespace.svc.cluster.local
```

4. Persistent Volume Claims (PVCs):

```
kubectl get pvc
```

The output is the following:

```
NAME                                    STATUS
VOLUME
                CAPACITY      ACCESS MODES   STORAGECLASS   AGE
tt-persistent-samplescaleout-data-1-0   Bound
ocid1.volume.oc1.phx.abyhqljtl63wgaxd3nvengilkaxs5h2b23kmtinpzmmqt7b
kzjdpnfu2c2fq   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-data-1-1   Bound
ocid1.volume.oc1.phx.abyhqljthxuwtpqx7rjtwvwsxjjkbgr25amjk7wtk26untb
qrealcqhe324q   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-data-1-2   Bound
ocid1.volume.oc1.phx.abyhqljthtm6frcjtttp6ye7hq4w5vm3jxyay54f4xtcbol
v2ercjeca5khq   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-data-2-0   Bound
ocid1.volume.oc1.phx.abyhqljtxgzm3raxj5sfoe56aonlh2mqqjre4quva4k3q3z
bbe7lftwqk3xa   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-data-2-1   Bound
ocid1.volume.oc1.phx.abyhqljtk3htdair4akll5dfrwpkipv3acjtww5hx3x2fz4
6af7zdew7gsiq   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-data-2-2   Bound
ocid1.volume.oc1.phx.abyhqljtiekgulpbadtwigehsuml75sngcuqjgqmx77lfpi
3wdeecbecavfa   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-mgmt-0     Bound
ocid1.volume.oc1.phx.abyhqljtacxwop7r2wqx6hsvun4haaydnco3y6g3rkgbwp5
hq35alay7uwaq   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-zk-0       Bound
ocid1.volume.oc1.phx.abyhqljtyoa5hdchax4sus652jtp665ckaef2cq3lakac2l
q52vfbls6kkcq   53687091200   RWO            oci            14m
tt-persistent-samplescaleout-zk-1       Bound
ocid1.volume.oc1.phx.abyhqljtongpcoggzpg2is25vmumijmah5gustwc3avgnij
rjigtqphtrana   53687091200   RWO            oci            13m
tt-persistent-samplescaleout-zk-2       Bound
ocid1.volume.oc1.phx.abyhqljttmgoljskb2ruawzv365uit7lsln2sbfno5e4vhh
6plbgh4tiblfq   53687091200   RWO            oci            10m
```

The Operator automatically creates one or two Persistent Volume Claims (PVCs) per Pod. These PVCs cause Persistent Volumes (PVs) to be allocated by Kubernetes and attached to the TimesTen Pods. TimesTen uses these PVs to hold the TimesTen instance and database. If you specify two PVCs, one PV holds the TimesTen instance and the checkpoint files and the second PV holds the

TimesTen transaction logs. In this example, the Operator creates one PVC for each Pod for a total of six PVCs. Each of the six PVs hold the TimesTen instance and an element of the database.

The Operator creates one PVC for the Pod that contains the management instance.

The Operator creates one PVC for each Pod that runs a ZooKeeper instance. This PVC causes a PV to be allocated by Kubernetes and attached to the ZooKeeper Pod. Each PV holds ZooKeeper's persistent data. In this example, since there are three ZooKeeper instances, the Operator created three PVCs.

Connect to the database

You can establish a shell in a TimesTen Pod and connect to the TimesTen database in the grid. You can then run operations in this TimesTen database.

1. Establish a shell in the TimesTen `samplescaleout-data-1-0` Pod.

   ```
   kubectl exec -it samplescaleout-data-1-0 -c tt -- /bin/bash
   ```

2. Connect to the `samplescaleout` database. Verify the information from the metadata files is in the database. This example connects to the database as the `sampleuser` user. (This user was created in the `schema.sql` file). The example then calls the `ttConfiguration` built-in procedure to check that the `PermSize` connection attribute has a value of `200` for this element of the database. The example then verifies the `sampleuser.emp` table exists.

   ```
   ttisql -connstr "DSN=samplescaleout;uid=sampleuser;pwd=sampleuserpwd";
   ```

   The output is similar to the following:

   ```
   Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights
   reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=samplescaleout;uid=sampleuser;pwd=********";
   Connection successful:
   DSN=samplescaleout;Durability=0;UID=sampleuser;DataStore=/tt/home/
   timesten/datastore/samplescaleout;
   DatabaseCharacterSet=AL32UTF8;PermSize=200;
   Connections=100;
   (Default setting AutoCommit=1)
   Command> call ttConfiguration ('PermSize');
   < PermSize, 200 >
   1 row found.
   Command> tables;
     SAMPLEUSER.EMP
   1 table found.
   ```

Manage the database

You can establish a shell in the TimesTen Pod of the management instance. You can then use the `ttGridAdmin` utility to manage and monitor the grid, including the health of the TimesTen database and its elements.

1. Establish a shell in the TimesTen `samplescaleout-management-0` Pod.

   ```
   kubectl exec -it samplescaleout-mgmt-0 -c tt -- /bin/bash
   ```

2. Run the `ttGridAdmin dbStatus -all` to check the status of the TimesTen database.

   ```
   ttGridAdmin dbStatus -all
   ```

   The output is the following:

   ```
   Database samplescaleout summary status as of Sun Jan 15 17:00:11
   UTC 2023

   created,loaded-complete,open
   Completely created elements: 6 (of 6)
   Completely loaded elements: 6 (of 6)
   Completely created replica sets: 3 (of 3)
   Completely loaded replica sets: 3 (of 3)

   Open elements: 6 (of 6)

   Database samplescaleout element level status as of Sun Jan 15
   17:00:11 UTC 2023

   Host                     Instance  Elem Status Cache Agent Date/Time
   of Event  Message
   ------------------------ --------- ---- ------ -----------
   ------------------- -------
   samplescaleout-data-1-0 instance1    1 opened stopped
   2023-01-15 16:28:57
   samplescaleout-data-1-1 instance1    2 opened stopped
   2023-01-15 16:28:57
   samplescaleout-data-1-2 instance1    3 opened stopped
   2023-01-15 16:28:56
   samplescaleout-data-2-0 instance1    4 opened stopped
   2023-01-15 16:28:57
   samplescaleout-data-2-1 instance1    5 opened stopped
   2023-01-15 16:28:56
   samplescaleout-data-2-2 instance1    6 opened stopped
   2023-01-15 16:28:56

   Database samplescaleout Replica Set status as of Sun Jan 15
   17:00:11 UTC 2023

   RS DS Elem Host                    Instance  Status Cache Agent
   Date/Time of Event  Message
   ```

```
-- -- ---- ----------------------- --------- ------ -----------
------------------- -------
 1  1    1 samplescaleout-data-1-0 instance1 opened stopped
2023-01-15 16:28:57
 1  2    4 samplescaleout-data-2-0 instance1 opened stopped
2023-01-15 16:28:57
 2  1    2 samplescaleout-data-1-1 instance1 opened stopped
2023-01-15 16:28:57
 2  2    5 samplescaleout-data-2-1 instance1 opened stopped
2023-01-15 16:28:56
 3  1    3 samplescaleout-data-1-2 instance1 opened stopped
2023-01-15 16:28:56
 3  2    6 samplescaleout-data-2-2 instance1 opened stopped
2023-01-15 16:28:56

Database samplescaleout Data Space Group status as of Sun Jan 15 17:00:11
UTC 2023

DS RS Elem Host                    Instance  Status Cache Agent Date/Time
of Event  Message
-- -- ---- ----------------------- --------- ------ -----------
------------------- -------
 1  1    1 samplescaleout-data-1-0 instance1 opened stopped
2023-01-15 16:28:57
 1  2    2 samplescaleout-data-1-1 instance1 opened stopped
2023-01-15 16:28:57
 1  3    3 samplescaleout-data-1-2 instance1 opened stopped
2023-01-15 16:28:56
 2  1    4 samplescaleout-data-2-0 instance1 opened stopped
2023-01-15 16:28:57
 2  2    5 samplescaleout-data-2-1 instance1 opened stopped
2023-01-15 16:28:56
 2  3    6 samplescaleout-data-2-2 instance1 opened stopped
2023-01-15 16:28:56
```

7

Using TimesTen Databases

This chapter explains how to use direct mode applications and Client/Server drivers to access and use TimesTen Classic databases in an active standby pair configuration and a TimesTen Scaleout database in the grid.

Topics:

- Using direct mode applications
- Using Client/Server drivers

Using direct mode applications

You can run direct mode applications inside of the Pods in your TimesTenClassic and TimesTenScaleout deployments.

When configured in your TimesTenClassic deployment, each Pod in your active standby pair runs two or more containers. When configured in your TimesTenScaleout deployment, each Pod containing a data instance runs two or more containers. One container in each Pod runs TimesTen and the TimesTen agent and the other container(s) run whatever applications you choose. The applications that are running in your containers can use TimesTen in direct mode. For information on direct mode applications, see "Managing TimesTen Databases" in the *Oracle TimesTen In-Memory Database Operations Guide*.

TimesTen Pods are created with the Kubernetes *shareProcessNamespace* option. This option allows direct mode applications running in other containers within the same Pod to function properly.

Note:

The standard security issues that surround direct mode apply in this environment as in a non-Kubernetes environment. Segregating your applications into separate containers from TimesTen is intended for ease of management and ease of upgrade. It is not intended as a security barrier and provides no additional security.

Use the `.spec.template.spec.containers` attribute of your TimesTenClassic object or the `.spec.dataTemplate.spec.containers` of your TimesTenScaleout object to cause one or more containers to be created within each of the TimesTen Pods that runs the `tt` container. Such containers are created in addition to the `tt` container that runs TimesTen.

This example illustrates how to include the `.spec.template.spec.containers` attribute in your TimesTenClassic object definition. For a TimesTenScaleout object, replace `template` with one or more `dataTemplate`s.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: directmode
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - directmode
  template:
    spec:
      containers:
      - name: yourapp
        image: phx.ocir.io/youraccount/yourapplication:2
        command: ["/bin/yourapp"]
      - name: anotherapp
        image: phx.ocir.io/youraccount/anotherapplication:2
        command: ["/bin/anotherapp"]
```

You can specify any other Kubernetes configuration for these containers.

The Operator automatically adds appropriate mounts to the containers. This gives your containers the ability to access TimesTen.

To use TimesTen in direct mode, your application containers must know how TimesTen is configured in the `tt` container. You must configure your application containers similarly.

In particular:

- You must know the name of the TimesTen users group. If you are using a container image located at `container-registry.oracle.com/timesten`, the name of the TimesTen users group is `timesten`. If you built the TimesTen container image, and changed the `timesten` default, ensure you use the name you used when you built the container image. See Option 2b: Build with customizations and Dockerfile ARGs.
- You must know the name of the Linux operating system user that runs TimesTen. The default is `timesten`. As was mentioned in the previous bullet, if you changed this default, ensure you use the name you used when you built the container image.
- You must configure your application containers to run your applications as a member of the TimesTen users group. Only members of this group can run TimesTen in direct mode.
- You can run your direct mode applications as a user with the same UID as that of the TimesTen user that runs TimesTen (`3429` is the default) . However, this grants the application instance administrator permissions on the TimesTen instance. Alternatively, you can create a group with the same GID as that of the TimesTen users group of and then create a user whose primary or secondary group is that group, but with a UID that is not the UID of the TimesTen user that is running TimesTen. In this case, you can run your application as this user and also use TimesTen in direct mode. You can then grant this user privileges up to and including the `ADMIN` privilege. For more information on primary and secondary groups, see "Creating an installation on Linux/UNIX" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*. For information on TimesTen privileges, see "System privileges" and "Object privileges" in the *Oracle TimesTen In-Memory Database SQL Reference*.

- The direct mode application must use the TimesTen instance that is configured at `/tt/home/timesten/instances/instance1`. The scripts to configure the TimesTen environment variables are located at `/tt/home/timesten/instances/instance1/bin/ttenv.*`.
- Do not modify any file that is located in the TimesTen instance. In addition, ensure you do not create any new files in the `$TIMESTEN_HOME` directory tree of the instance.
- Do not add entries to the `/tt/home/timesten/instances/instance1/conf/sys.odbc.ini` file. These files can be overwritten by the Operator. However, you can store your own DSN entries in the `$HOME/.odbc.ini` file located in your application container.
- Do not create additional TimesTen databases.

Kubernetes, not the Operator, is responsible for monitoring and managing the life cycle of the direct mode containers. In particular:

- Applications are started by Kubernetes regardless of the state of TimesTen (located in its own container). Kubernetes manages the life cycle of containers individually. It does not sequence. Your application must know how to wait for TimesTen to become available.
- For a TimesTenClassic object, a direct mode application runs in the Pod containing the active TimesTen database and in the Pod containing the standby TimesTen database. The application may need to use the `ttRepStateGet` built-in procedure to determine whether it is running on the active or on the standby and perhaps quiesce itself on the standby. For more information on the `ttRepStateGet` built-in procedure, see "ttRepStateGet" in the *Oracle TimesTen In-Memory Database Reference*.
- Kubernetes may start the application before the TimesTen database exists or before it is loaded into memory and ready for use. It is the responsibility of the direct mode application to verify the state of the TimesTen database in its Pod and to use it appropriately.
- If your application exits, the container terminates, and Kubernetes spawns another container. This does not impact TimesTen that is running in the `tt` container.

Using Client/Server drivers

Applications that are running in other Pods in your Kubernetes cluster can use your TimesTen database by using the standard TimesTen Client/Server drivers. You must configure your application containers with a TimesTen client instance. That instance must contain a configured *`$TIMESTEN_HOME`*`/conf/sys.odbc.ini` file, or your application must use an appropriate Client/Server connection string.

In TimesTen Classic, if you chose to configure a `sys.odbc.ini` file, the contents of `sys.odbc.ini` contains a client DSN definition that references the Pods that are running your TimesTen databases. In TimesTen Scaleout, you can export a `sys.odbc.ini` file for use by client/server clients outside of the grid. Use the `ttGridAdmin` utility with the `gridClientExport` or `gridClientExportAll` for this purpose. See ttGridAdmin in the *Oracle TimesTen In-Memory Database Reference*.

This example creates the `sample` DSN and references the `sample` TimesTenClassic object in the `mynamespace` namespace.

```
% vi $TIMESTEN_HOME/conf/sys.odbc.ini

[sample]
```

```
TTC_SERVER_DSN=sample
TTC_SERVER1=sample-0.sample.mynamespace.svc.cluster.local
TTC_SERVER2=sample-1.sample.mynamespace.svc.cluster.local
```

Applications connect to the TimesTen database using this DSN. In the TimesTen Classic active standby pair configuration, TimesTen automatically routes application connections to the active database. (`sample-0` and `sample-1` are used for example purposes.)

Client/Server applications must connect to the database using a defined username and password. The Operator can create such a user with `ADMIN` privileges. You can then connect to the database, as that user, to create other users and grant these users the `CREATE SESSION` privilege. See Overview of the configuration metadata and the Kubernetes facilities.

In this example, use a connection string to connect to the `sample` database as the `sampleuser` user. (If you use a connection string that requires all the required connection attributes, you do not need to define them in the `sys.odbc.ini` file.) The `sampleuser` user was created by the Operator and already exists in the `sample` database. After connecting, you can verify that the `sampleuser.emp` table exists. (The Operator also previously created this table. See schema.sql for information on how the Operator created this table.)

```
% ttIsqlCS -connstr "TTC_SERVER1=sample-0.sample.mynamespace.svc.cluster.local;
TTC_SERVER2=sample-1.sample.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=sample;UID=sampleuser;PWD=samplepw";


Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights
reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "TTC_SERVER1=sample-0.sample.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=sample;uid=sampleuser;pwd=********";
Connection successful:
DSN=;TTC_SERVER=sample-0.sample.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=sample;UID=sampleuser;DATASTORE=/tt/home/timesten/datastore/
sample;
DATABASECHARACTERSET=AL32UTF8;PERMSIZE=200;
DDLREPLICATIONLEVEL=3;
(Default setting AutoCommit=1)
Command> tables;
  SAMPLEUSER.EMP
1 table found.
```

8

Managing and Monitoring Your Active Standby Pairs

This chapter discusses how to monitor the health of each Pod in your active standby pair as well as the health of the active standby pairs themselves. It details the `BothDown` and the `ManualInterventionRequired` states with an emphasis of how the Operator behaves in each of these states. The chapter discusses how to suspend the management of your TimesTenClassic object by the Operator. It concludes with various manual operations you can perform on your TimesTen databases.

Topics:

Monitoring the health of each Pod in the active standby pair

The Operator keeps track of the individual health and state of each Pod in the active standby pair. How often the Operator checks the health is defined by the value of the `pollingInterval`. See "TimesTenClassicSpecSpec" for information on `pollingInterval`.

Each Pod is assigned a high level state based on the state of various components of Kubernetes and the state of TimesTen. These states are:

- CatchingUp
- Down
- Healthy
- HealthyActive
- HealthyStandby
- OtherDown
- Terminal
- Unknown
- UpgradeFailed

CatchingUp

The standby has completed the process of duplicating the database from the active. The newly created standby is catching up to any transactions that ran on the active while the duplicate operation was running.

Down

Either the Pod or the TimesTen components within the Pod (or both) are not functioning properly, given this Pod's role in the active standby pair.

Healthy

The Pod and the TimesTen components within the Pod are in a healthy state, given this Pod's role in the active standby pair.

HealthyActive

When a TimesTenClassic object is in the `Reexamine` state, the Operator examines the state of both TimesTen instances. The Operator does not know which instance (if any) contains a properly configured active database (or a properly configured standby database). The Operator must examine both instances to see. If a healthy instance is found and that instance contains a properly configured active database, the state of the Pod is reported as `HealthyActive`.

HealthyStandby

When a TimesTenClassic object is in the `Reexamine` state, the Operator examines the state of both TimesTen instances. The Operator does not know which instance (if any) contains a properly configured standby database (or a properly configured active database). The Operator must examine both instances to see. If a healthy instance is found and that instance contains a properly configured standby database, the state of the Pod is reported as `HealthyStandby`.

OtherDown

The Pod and the TimesTen components within the Pod are in a healthy state, but TimesTen in this Pod believes that TimesTen in the other Pod has failed. In particular, the `OtherDown` state indicates that this Pod contains an active database, and the database's peer has reached the `failThreshold`. The database in this Pod is no longer keeping transaction logs for its peer, as the peer is too far behind. Recovering the peer requires re-duplicating the active database (which the Operator will perform automatically).

Terminal

TimesTen in the Pod cannot be repaired by the Operator.

Unknown

The state of this Pod is unknown. Either the Pod is unreachable or the TimesTen agent contained within the Pod has failed.

UpgradeFailed

An automated upgrade was attempted on TimesTen in this Pod and the upgrade failed. See About upgrading TimesTen Classic.

Monitoring the health of the active standby pair of databases

The Operator monitors and manages the health of each of your active standby pairs. The Operator assigns high level states to the TimesTenClassic object, which you can monitor and review. For example, you can use the `kubectl get` command to return the high level state of your TimesTenClassic object. Specifically, in this example, the value returned for the `STATE` field is `Normal`, indicating that the active and the standby databases are up and running, and working as they should.

```
% kubectl get ttc sample
NAME      STATE     ACTIVE     AGE
sample    Normal    sample-0   15h
```

The states:

- ActiveDown
- ActiveTakeover
- BothDown
- ConfiguringActive
- Failed
- Initializing
- ManualInterventionRequired
- Normal
- Reexamine
- StandbyCatchup
- StandbyDown
- StandbyStarting
- WaitingForActive

ActiveDown

If the Operator detects that TimesTen in the Pod containing the active database has failed, then the TimesTenClassic object immediately enters the `ActiveDown` state.

The `unreachableTimeout` timeout value controls how long the state of the Pod containing the active database can be `Unknown` before the TimesTenClassic object's state becomes `ActiveDown`.

When the TimesTenClassic object's state becomes `ActiveDown`, the standby database immediately becomes the active, and the state of the TimesTenClassic object becomes `StandbyDown`.

ActiveTakeover

When the TimesTenClassic object is in the `Normal` state, and the standby database goes down, the state briefly changes to `ActiveTakeover`.

When AWT cache groups are used, the standby is normally responsible for pushing updates from TimesTen to Oracle Database. However, if the standby fails, the active database takes over this responsibility. This occurs during the `ActiveTakeover` state.

BothDown

Neither the active nor the standby database is functioning properly. The Operator attempts to bring up the pair of databases.

If both Pods in the active standby pair fail, the Operator uses the information in TimesTenClassicStatus to minimize data loss. See "Understanding the BothDown state" for details.

ConfiguringActive

When the TimesTenClassic object is in the `WaitingForActive` state, and when the database that should be the active database comes up, the TimesTenClassic object enters the `ConfiguringActive` state. The Operator then configures this database to be the active. Once the database is configured as the active, the TimesTenClassic object enters the `StandbyDown` state. See "Understanding the BothDown state" for details.

Failed

If a problem occurs while `Initializing` a TimesTenClassic object, the object transitions to the `Failed` state. Once in this state, the Operator does not attempt to repair the object. You must delete it. Use the `kubectl describe` command to examine the Operator logs to determine the cause of the problem and then recreate the object.

Initializing

This state is reported while the two Pods are starting up for the first time. In your active standby pair configuration, the Pod whose name ends with `-0` is initially configured as the active database, and the Pod whose name ends with `-1` is initially configured as the standby database. Specifically, if you specified the name for TimesTenClassic as `sample`, the `sample-0` Pod is configured as the active database, and the `sample-1` Pod is configured as the standby database. Once the active/standby pair is completely deployed, the TimesTenClassic object transitions to the `Normal` state.

ManualInterventionRequired

When a TimesTenClassic object enters the `ManualInterventionRequired` state, the Operator takes no further action for the object. It does not query the TimesTen agents associated with the object to determine the state of TimesTen and it does not command TimesTen to do anything. See "Understanding the ManualInterventionRequired state" and "Bringing up one database" for details.

Normal

Both databases are up and running, and operating as they should.

Reexamine

When the TimesTenClassic object is in the `ManualInterventionRequired` state, you can specify the `.spec.ttspec.reexamine` datum to cause the Operator to take over the management of the object again. The Operator moves the object to the `Reexamine` state. The Operator then examines the state of TimesTen. If you correctly repaired TimesTen, the TimesTenClassic object may then enter the `Normal` or the `StandbyDown` state, depending on the nature of your repair. If you did not correctly repair TimesTen, the TimesTenClassic object re-enters the `ManualInterventionRequired` state. See "Understanding the ManualInterventionRequired state" for details.

StandbyCatchup

This state is entered after the `StandbyStarting` state. During the `StandbyStarting` state, the standby copies the active database to the standby Pod. When the duplicate process is complete, the state changes from `StandbyStarting` to `StandbyCatchup`. See "StandbyStarting" for more information on the `StandbyStarting` state. In the `StandbyCatchup` state, the duplicate process has completed. Transactions that ran during this duplicate process must now be copied over to the standby. Thus the `StandbyCatchup` state is the state when the newly created standby catches up to any transactions that ran on the active while the duplicate operation was running. Applications can continue to use the active without restriction.

StandbyDown

The active database is functioning properly, but the standby database is not. The Operator automatically attempts to restart and reconfigure the standby database. Applications can continue to use the active database without restriction.

StandbyStarting

The standby is duplicating the database from the active. The `StandbyStarting` state is complete when the duplicate operation completes. The `StandbyCatchup` state is then entered. See "StandbyCatchup" for more information on the `StandbyCatchup` state. Applications can continue to use the active without restriction.

WaitingForActive

When the TimesTenClassic object is in the `BothDown` state, if the Operator can determine which database contains the most up-to-date data, the TimesTenClassic object enters the `WaitingForActive` state. The object remains in this state until the Pod that contains the database is running, and the TimesTen agent within the `tt` container (within that Pod) is responding to the Operator. See "Understanding the BothDown state" for details.

Understanding the BothDown state

The Operator provisions, monitors, and manages active standby pairs of TimesTen databases. It detects and reacts to the failure of the active or the standby database. For example, when one database in the active standby pair is down, the Operator does the following:

- If the active database fails, the Operator promotes the standby to be the active.
- If the standby database fails, the Operator keeps the active running and repairs the standby.

However, if both databases fail at the same time, it is essential that the databases are brought back up appropriately. TimesTen replication does not atomically commit transactions in both database simultaneously. Transactions are committed in one database and then later are committed in the other database. (The database on which transactions are committed first is considered the database that is *ahead*.) Depending on how replication is configured, transactions on the active database may be ahead of the standby or the standby may be ahead of the active. To avoid data loss, the database that is ahead must become the active database after the failure is corrected.

In most cases, the Operator can determine which database was ahead at the time of the failure. However, there are cases where the Operator cannot determine which database was ahead. In particular, the Operator cannot determine which database is ahead if all of the following conditions occur:

- Both databases failed during the polling interval. Specifically, the Operator examined both databases and the TimesTen Pods were in the `Healthy` state. The Operator waited `pollingInterval` seconds, and when the Operator examined the databases again (after this `pollingInterval`), both databases were down **and**
- `RETURN TWOSAFE` replication was configured **and**
- `DISABLE RETURN` or `LOCAL COMMIT ACTION COMMIT` (or both) were configured.

See TimesTenClassicSpecSpec for more information on the `.spec.ttspec.pollingInterval` datum and on the `RETURN TWOSAFE` and `DISABLE RETURN` replication configurations options. Also, see CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* and Defining an active standby pair replication scheme in the *Oracle TimesTen In-Memory Database Replication Guide* for information on defining an active standby pair replication scheme.

This combination of events indicates that some transactions may have committed on the standby and not on the active and/or some transactions may have committed on the active and not on the standby. The Operator takes no action in this case.

When both databases fail, the TimesTenClassic object enters the `BothDown` state. See BothDown for more information on the `BothDown` state. The Operator must then determine the appropriate action to take. The Operator first examines the value of the `.spec.ttspec.bothDownBehavior` datum to determine what to do. See TimesTenClassicSpecSpec.

If `.spec.ttspec.bothDownBehavior` is set to `Manual`, the TimesTenClassic object immediately enters the `ManualInterventionRequired` state. The Operator takes no further action even if either TimesTen container subsequently becomes available. See

Understanding the ManualInterventionRequired state for information on the `ManualInterventionRequired` state.

If `.spec.ttspec.bothDownBehavior` is set to `Best` (the default setting), the Operator attempts to determine which database was ahead at the time of failure.

- If the Operator cannot determine which database is ahead, the TimesTenClassic object immediately enters the `ManualInterventionRequired` state. See Understanding the ManualInterventionRequired state.
- If the Operator can determine which database is ahead:
  - The TimesTenClassic object enters the `WaitingForActive` state. The object remains in this state until the Pod containing that database is running and the TimesTen agent located in the `tt` container within that Pod is responding to the Operator. At this point, the TimesTenClassic object enters the `ConfiguringActive` state.
  - While the TimesTenClassic object is in the `ConfiguringActive` state, TimesTen in this Pod is started, the database is loaded and is configured for use as the new active database. If there are any problems with these steps, the TimesTenClassic object enters the `ManualInterventionRequired` state. If the database is successfully loaded and successfully configured as the new active, the TimesTenClassic object enters the `StandbyDown` state. See Monitoring the health of the active standby pair of databases for information on the states of your TimesTenClassic object.
  - You can specify the maximum amount of time (expressed in seconds) that the TimesTenClassic object remains in the `WaitingForActive` state by specifying a value for the `spec.ttspec.waitingForActiveTimeout` datum. After this period of time, if the object is still in the `WaitingForActive` state, the object automatically transitions to the `ManualInterventionRequired` state. The default is `0`, which indicates that there is no timeout, and the object will remain in this state indefinitely. See TimesTenClassicSpecSpec for more information on the `spec.ttspec.waitingForActiveTimeout` datum.
  - The time to recover the database varies by the size of the database. You should consider the size of your database when deciding the value for `spec.ttspec.waitingForActiveTimeout`.
  - If the database that is ahead cannot be loaded, the TimesTenClassic object enters the `ManualInterventionRequired` state. See Understanding the ManualInterventionRequired state.

Understanding the ManualInterventionRequired state

When a TimesTenClassic object enters the `ManualInterventionRequired` state, the Operator takes no further action for this object. It does not query the TimesTen agents associated with the object to determine the state of TimesTen and does not command TimesTen to do anything. It is important for you to address why the TimesTenClassic object is in this state.

If your TimesTenClassic object is in the `ManualInterventionRequired` state and it is not the result of it first being in the `BothDown` state, perform the operations necessary to manually repair one of the databases. Then, perform the steps to bring up this database. These steps are covered in "Bringing up one database" later in this chapter.

If, however, the TimesTenClassic object is in the `ManualInterventionRequired` state as a result of it first being in the `BothDown` state:

- It may be unclear which database, if either, is suitable to be the new active. There may be transactions that have committed on the active database and not on the standby database, and simultaneously there may be transactions that have committed on the standby database and not on the active database.
- You need to manually examine both databases and may need to reconcile the data before you can choose which database should be the new active.
- If you can reconcile the data, and can manually fix one of the databases, then you can perform the steps to bring up one database. These steps are covered in "Bringing up one database" later in this chapter. If you cannot reconcile the data, contact Oracle Support for further assistance.

In order for you to direct the Operator to move the TimesTenClassic object out of the `ManualInterventionRequired` state, you must either:

- Bring up exactly one database: The Operator treats this database as the active database. All of these conditions must be met:
  - The TimesTen agent in the container is running.
  - The TimesTen the instance in the container is running.
  - The TimesTen database is loaded.
  - There is no replication scheme in the database.
  - The replication agent is not running.
  - The replication state is `IDLE`.

  If these conditions are met, the Operator moves the TimesTenClassic object to the `StandbyDown` state. If any of these conditions are not met, the TimesTenClassic object remains in the `ManualInterventionRequired` state. Note that when no replication scheme exists in the database, the Operator will still create the appropriate replication scheme based on how it is defined in the TimesTenClassic object definition. See "Bringing up one database" for an example of how you can direct the Operator to take action once one database is up and running.
- Bring up both databases: In this case, you must configure the active standby pair. Specifically, each database must meet all of the following conditions:
  - The TimesTen agent in the container is running.
  - The TimesTen instance in the container is running.
  - The database is loaded.
  - The replication scheme is defined in both databases.
  - The replication agents are started and are running.
  - One database must be in the `ACTIVE` state and the other database must be in the `STANDBY` state.

  If these conditions are met, the Operator moves the TimesTenClassic object to the `Normal` state. If any of these conditions are not met, the TimesTenClassic object remains in the `ManualInterventionRequired` state.

If you cannot bring up either database, the TimesTenClassic object remains in the `ManualInterventionRequired` state.

You direct the Operator to examine the databases by specifying the `.spec.ttspec.reexamine` datum. Every `.spec.ttspec.pollingInterval`, the

Operator examines the value of `.spec.ttspec.reexamine`. If the value has changed since the last iteration for this TimesTenClassic object, the Operator examines the state of the TimesTen containers for this object. See "TimesTenClassicSpecSpec" for more information on the `.spec.ttspec.pollingInterval` and the `.spec.ttspec.reexamine` datum.

The examination of the databases is performed exactly one time after you change the `.spec.ttspec.reexamine` value. If the required conditions were not met, you may again attempt to meet them. You must then modify the `.spec.ttspec.reexamine` value again to cause the Operator to reexamine the databases.

Note that whenever a TimesTenClassic object changes state, a Kubernetes Event is created. You can monitor these events with the `kubectl describe` command to be informed of such state transitions.

Bringing up one database

This section assumes you have manually repaired or have manually performed maintenance on one of the databases associated with the TimesTenClassic object. The TimesTenClassic object is currently in the `ManualInterventionRequired` state. You now want to direct the Operator to treat the repaired database as the active, to perform the necessary steps to duplicate this database to the standby, and to bring up both databases, such that both are running and operating successfully.

Recall that all of these conditions must be met for the database:

- TimesTen agent in the container is running.
- TimesTen daemon (the instance) in the container is running.
- TimesTen database is loaded.
- There is no replication scheme in the database.
- The replication agent is not running.
- The replication state is `IDLE`.

These sections show you how to verify the conditions are met for the database and how to set the `reexamine` value:

- Verify the conditions are met for the database
- Set the reexamine value

Verify the conditions are met for the database

Perform these steps to ensure the conditions are met for the database (the database to be the active). In this example, `sample-1` will be the new active.

Note: These steps require you to use TimesTen utilities and TimesTen built-in procedures. See Utilities and Built-In Procedures in the *Oracle TimesTen In-Memory Database Reference* for details.

1. Confirm the TimesTenClassic object (`sample`, in this example) is in the `ManualInterventionRequired` state (represented in **bold**).

```
% kubectl get ttc sample
NAME     STATE                        ACTIVE     AGE
sample   ManualInterventionRequired   sample-0   12h
```

2. Use the `kubectl exec -it` command to invoke the shell within the `sample-1` Pod that contains the TimesTen database. (This database will be the new active.)

   The remaining procedures take place within this shell.

   ```
   % kubectl exec -it sample-1  -c tt -- /bin/bash
   ```

3. Use the `ttDaemonAdmin` utility to start TimesTen daemon (if not already started). Then use the `ttAdmin` utility to load the TimesTen database into memory (if not already loaded).

   ```
   % ttDaemonAdmin -start
   TimesTen Daemon (PID: 5948, port: 6624) startup OK.
   % ttAdmin -ramLoad sample
   RAM Residence Policy            : manual
   Manually Loaded In RAM          : True
   Replication Agent Policy        : manual
   Replication Manually Started    : False
   Cache Agent Policy              : manual
   Cache Agent Manually Started    : False
   Database State                  : Open
   ```

4. Use the `ttIsql` utility to connect to the `sample` database. Then, call the `ttRepStop` built-in procedure to stop the replication agent.

   ```
   % ttIsql sample

   Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=sample";
   Connection successful: DSN=sample;UID=timesten;DataStore=/tt/home/timesten/
   datastore/sample;
   DatabaseCharacterSet=AL32UTF8;AutoCreate=0;
   PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=1;
   (Default setting AutoCommit=1)
   Command> call ttRepStop;
   ```

5. From within `ttIsql`, use the SQL `DROP ACTIVE STANDBY PAIR` statement to drop the active standby pair replication scheme. Then use the `ttIsql repschemes` command to verify there are no replication schemes in the database. Exit from `ttIsql`.

   ```
   Command> DROP ACTIVE STANDBY PAIR;
   Command> repschemes;

   0 replication schemes found.
   ```

6. Use the `ttStatus` utility to verify the TimesTen daemon is running and the replication agent is not running.

   ```
   % ttStatus
   TimesTen status report as of Sat Apr 24 02:14:15 2022

   Daemon pid 5948 port 6624 instance instance1
   TimesTen server pid 5955 started on port 6625
   ------------------------------------------------------------------------
   ------------------------------------------------------------------------
   Data store /tt/home/timesten/datastore/sample
   Daemon pid 5948 port 6624 instance instance1
   TimesTen server pid 5955 started on port 6625
   ```

```
There are 15 connections to the data store
Shared Memory KEY 0x0a100c60 ID 196609
PL/SQL Memory Key 0x0b100c60 ID 229378 Address 0x5000000000
Type            PID     Context             Connection Name              ConnID
Process         10418   0x000000000218a6e0  sample                            2
Process         8338    0x0000000001cbb6e0  sample                            1
Subdaemon       5953    0x00000000015075f0  Manager                        2047
Subdaemon       5953    0x0000000001588540  Rollback                       2046
Subdaemon       5953    0x0000000001607210  Checkpoint                     2041
Subdaemon       5953    0x00007f132c0008c0  Flusher                        2045
Subdaemon       5953    0x00007f132c080370  Log Marker                     2040
Subdaemon       5953    0x00007f13340008c0  Monitor                        2044
Subdaemon       5953    0x00007f133407f330  HistGC                         2037
Subdaemon       5953    0x00007f13380008c0  Aging                          2042
Subdaemon       5953    0x00007f133807f330  AsyncMV                        2039
Subdaemon       5953    0x00007f133c0008c0  Deadlock Detector              2043
Subdaemon       5953    0x00007f133c07f330  IndexGC                        2038
Subdaemon       5953    0x00007f135c0008c0  Garbage Collector              2035
Subdaemon       5953    0x00007f13600e8e20  XactId Rollback                2036
Open for user connections
RAM residence policy: Manual
Data store is manually loaded into RAM
Replication policy  : Manual
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group timesten
End of report
```

You have successfully verified the conditions for the database. The database is up and running. The Operator will treat this database as the active. You are now ready to set the value for the `.spec.ttspec.reexamine` datum.

Set the reexamine value

This example shows you how to set the `reexamine` value in the TimesTenClassic object definition (`sample`, in this example). The example also illustrates the action the Operator takes after the `reexamine` value has been changed.

1. Set the `reexamine` value. The value must be different than the current value for the TimesTenClassic object. When the Operator examines this value and notices it has changed since the last iteration, it will take appropriate action.

   Use the `kubectl edit` command to edit the TimesTenClassic object.

   - If there is a line for `reexamine` in the file, then modify its value. It must be different than the current value.
   - If there is no line for `reexamine` in the file, then add a line and specify a value.

   In this example, there is no `reexamine` line. This example adds the `reexamine` line and sets the value for `reexamine` to `April22reexamine1` (represented in **bold**).

   Note: Not all output is shown.

   ```
   % kubectl edit timestenclassic sample
   # Please edit the object below. Lines beginning with a '#' will be ignored,
   # and an empty file will abort the edit. If an error occurs while saving this
   # file will be reopened with the relevant failures.
   #
   apiVersion: timesten.oracle.com/v1
   ```

```
kind: TimesTenClassic
metadata:
...
  name: sample
  namespace: mynamespace
...
repCreateStatement: |
  create active standby pair
    "{{tt-name}}" on "{{tt-node-0}}",
    "{{tt-name}}" on "{{tt-node-1}}"
  RETURN TWOSAFE
  store "{{tt-name}}" on "{{tt-node-0}}"
    PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999
  store "{{tt-name}}" on "{{tt-node-1}}"
    PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999
spec:
  ttspec:
    bothDownBehavior: Best
    dbConfigMap:
    - sample
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    storageClassName: oci
    storageSize: 250Gi
    reexamine: April22reexamine1
...
timestenclassic.timesten.oracle.com/sample edited
```

2. Use the `kubectl get` command to assess the state of the `sample` TimesTenClassic object. Observe how the state changes as you issue multiple `kubectl get` commands. Also note that the Operator has successfully configured `sample-1` to be the active.

```
% kubectl get ttc sample
NAME     STATE       ACTIVE   AGE
sample   Reexamine   None     68m
% kubectl get ttc sample
NAME     STATE               ACTIVE   AGE
sample   ConfiguringActive   None     68m
% kubectl get ttc sample
NAME     STATE         ACTIVE     AGE
sample   StandbyDown   sample-1   68m
% kubectl get ttc sample
NAME     STATE    ACTIVE     AGE
sample   Normal   sample-1   71m
```

3. Use the `kubectl describe` command to further review the actions of the Operator (represented in **bold**).

   Not all output is shown:

```
% kubectl describe ttc sample
Name:         sample
Namespace:    mynamespace
...
Kind:         TimesTenClassic
...
Rep Create Statement:  create active standby pair
  "{{tt-name}}" on "{{tt-node-0}}",
  "{{tt-name}}" on "{{tt-node-1}}"
RETURN TWOSAFE
store "{{tt-name}}" on "{{tt-node-0}}"
```

```
  PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999
store "{{tt-name}}" on "{{tt-node-1}}"
  PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999

Spec:
  Ttspec:
    Both Down Behavior:  Best
    Db Config Map:
      sample
    Image:               container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Reexamine:           April22reexamine1
    Stop Managing:       April21Stop1
    Storage Class Name:  oci
    Storage Size:        250Gi
Status:
  Classic Upgrade Status:
    Active Start Time:           0
    Active Status:
    Image Update Pending:        false
    Last Upgrade State Switch:   0
    Prev Reset Upgrade State:
    Prev Upgrade State:
    Standby Start Time:          0
    Standby Status:
    Upgrade Start Time:          0
    Upgrade State:
  Active Pods:                   sample-1
  High Level State:              Normal
  Last Event:                    54
  Last High Level State Switch:  1619230912
  Pod Status:
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:                          Loaded
      Db Id:                       475
      Db Updatable:                No
    Initialized:                   true
    Last High Level State Switch:  ?
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1619231126
      Pod IP:               10.244.7.89
      Pod Phase:            Running
    Prev High Level State:  Healthy
    Prev Image:
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Times Ten Status:
      Daemon:          Up
      Instance:        Exists
      Release:         22.1.1.9.0
    Admin User File:   false
```

```
      Cache User File:   false
      Cg File:           false
      Disable Return:    false
      High Level State:  Healthy
      Intended State:    Standby
      Local Commit:      false
      Name:              sample-0
      Schema File:       false
      Using Twosafe:     false
      Cache Status:
        Cache Agent:        Not Running
        Cache UID Pwd Set:  true
        N Cache Groups:     0
      Db Status:
        Db:                          Loaded
        Db Id:                       476
        Db Updatable:                Yes
      Initialized:                   true
      Last High Level State Switch:  ?
      Pod Status:
        Agent:                Up
        Last Time Reachable:  1619231126
        Pod IP:               10.244.6.149
        Pod Phase:            Running
      Prev High Level State:  Healthy
      Prev Image:
      Replication Status:
        Last Time Rep State Changed:  1619228670
        Rep Agent:                    Running
        Rep Peer P State:             start
        Rep Scheme:                   Exists
        Rep State:                    ACTIVE
      Times Ten Status:
        Daemon:             Up
        Instance:           Exists
        Release:            22.1.1.9.0
      Admin User File:      false
      Cache User File:      false
      Cg File:              false
      Disable Return:       false
      High Level State:     Healthy
      Intended State:       Active
      Local Commit:         false
      Name:                 sample-1
      Schema File:          false
      Using Twosafe:        false
    Prev High Level State:  StandbyDown
    Prev Reexamine:         April22reexamine1
    Prev Stop Managing:     April21Stop1
    Rep Create Statement:   create active standby pair "sample" on
"sample-0.sample.mynamespace.svc.cluster.local", "sample" on
"sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store "sample" on
"sample-0.sample.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
store
"sample" on "sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD 0
    Rep Port:               4444
    Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     StateChange  58m    ttclassic  TimesTenClassic was Normal, now
ManualInterventionRequired
  -     StateChange  46m    ttclassic  Pod sample-0 Daemon Down
  -     StateChange  41m    ttclassic  Pod sample-1 Daemon Down
  -     StateChange  41m    ttclassic  Pod sample-1 Daemon Up
  -     StateChange  41m    ttclassic  Pod sample-1 Database Unloaded
  -     StateChange  40m    ttclassic  Pod sample-1 Database Loaded
  -     StateChange  40m    ttclassic  Pod sample-1 RepState IDLE
  -     StateChange  40m    ttclassic  Pod sample-1 RepAgent Not Running
  -     StateChange  17m    ttclassic  Pod sample-1 Database Updatable
  -     StateChange  17m    ttclassic  Pod sample-1 RepScheme None
  -     StateChange  4m21s  ttclassic  TimesTenClassic was
ManualInterventionRequired, now Reexamine
  -     Error        4m16s  ttclassic  Active error: Daemon Down
  -     StateChange  4m16s  ttclassic  TimesTenClassic was Reexamine, now
ConfiguringActive
  -     StateChange  4m10s  ttclassic  Pod sample-1 RepState ACTIVE
  -     StateChange  4m10s  ttclassic  Pod sample-1 RepScheme Exists
  -     StateChange  4m10s  ttclassic  Pod sample-1 RepAgent Running
  -     StateChange  4m8s   ttclassic  TimesTenClassic was ConfiguringActive, now
StandbyDown
  -     StateChange  4m3s   ttclassic  Pod sample-0 Daemon Up
  -     StateChange  4m3s   ttclassic  Pod sample-0 Database Unloaded
  -     StateChange  3m56s  ttclassic  Pod sample-0 Database None
  -     StateChange  3m42s  ttclassic  Pod sample-0 Database Loaded
  -     StateChange  3m42s  ttclassic  Pod sample-0 Database Not Updatable
  -     StateChange  3m42s  ttclassic  Pod sample-0 RepAgent Not Running
  -     StateChange  3m42s  ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  3m36s  ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  3m36s  ttclassic  Pod sample-0 RepState STANDBY
  -     StateChange  3m36s  ttclassic  TimesTenClassic was StandbyDown, now Normal
```

4. Use the `kubectl exec -it` command to invoke the shell within the `sample-1` Pod that contains the TimesTen database. Then, verify you can connect to the active database.

```
% kubectl exec -it sample-1 -c tt -- /bin/bash
$ ttIsql sample

Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful: DSN=sample;UID=timesten;DataStore=/tt/home/timesten/
datastore/sample;
DatabaseCharacterSet=AL32UTF8;AutoCreate=0;PermSize=200;DDLReplicationLevel=3;Force
DisconnectEnabled=1;
(Default setting AutoCommit=1)
Command> call ttRepStateGet;
< ACTIVE >
1 row found.
```

5. Use the `kubectl exec -it` command to invoke the shell within the `sample-0` Pod that contains the TimesTen database. Then, verify you can connect to the standby database.

```
% kubectl exec -it sample-0 -c tt -- /bin/bash
% ttIsql sample

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
```

```
connect "DSN=sample";
Connection successful: DSN=sample;UID=timesten;DataStore=/tt/home/timesten/
datastore/sample;
DatabaseCharacterSet=AL32UTF8;AutoCreate=0;
PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)
Command> call ttRepStateGet;
< STANDBY >
1 row found.
```

The Operator is now managing and monitoring your TimesTenClassic object. The TimesTenClassic object is in the `Normal` state. Both databases are up and running and ready for use.

Suspending the management of a TimesTenClassic object

These sections discuss why you may want to suspend the management of your TimesTenClassic object by the Operator and then how to do it:

- Overview
- Suspend management of the TimesTenClassic object

Overview

The Operator periodically examines the state of the TimesTen instances and the databases associated with each TimesTenClassic object. It takes actions to repair anything that is broken. You may have a situation in which you want to manually perform maintenance operations. In such a situation, you do not want the Operator to interfere and attempt to perform repair operations.

You could stop the Operator (by deleting the Deployment of the `timesten-operator`). This action prevents the Operator from interfering. See "Revert to manual control" for more information. However, if you have more than one TimesTenClassic object and you delete the Operator, this interferes with the management of all the TimesTenClassic objects, when perhaps only one of them needs manual intervention.

Alternatively, you can direct the Operator to take no action for one TimesTenClassic object by specifying the `.spec.ttspec.stopManaging` datum for this TimesTenClassic object. See "TimesTenClassicSpecSpec" for more information on this element. The Operator examines the value of `.spec.ttspec.stopManaging` and if it has changed since the last time the Operator examined it, the Operator changes the state of the TimesTenClassic object to `ManualInterventionRequired`. This causes the Operator to no longer examine the status of the TimesTen Pods, the containers, the instances, and the databases associated with the TimesTenClassic object. The Operator takes no action on the object or its Pods.

When you want the Operator to manage the TimesTenClassic object again, you change the value of the `.spec.ttspec.reexamine` datum. See "Understanding the ManualInterventionRequired state" for more information on the `ManualInterventionRequired` state and the `.spec.ttspec.reexamine` datum.

In this way, you can perform manual operations on TimesTen without deleting the Deployment of the `timesten-operator`.

Suspend management of the TimesTenClassic object

This example illustrates how to use the `.spec.ttspec.stopManaging` datum to direct the Operator to stop managing one of the TimesTenClassic objects running in your Kubernetes cluster. In this example, there are two TimesTenClassic objects (`sample` and `sample2`) that are running. There is a requirement for you to perform manual maintenance operations on the TimesTen databases associated with one of the objects (`sample`, in this example). You want the Operator to stop managing this `sample` TimesTenClassic object. However, you want the Operator to continue managing the other TimesTenClassic object (`sample2`, in this example).

Perform these steps:

1. Review the Pods that are running.

```
% kubectl get pods
NAME                                         READY   STATUS    RESTARTS   AGE
sample-0                                     2/2     Running   0          6m33s
sample-1                                     2/2     Running   0          6m32s
sample2-0                                    2/2     Running   0          6m32s
sample2-1                                    2/2     Running   0          6m32s
timesten-operator-846cb5c97c-cxbl2           1/1     Running   0          4d20h
```

2. Confirm the `sample` TimesTenClassic object is in the `Normal` state. Recall that you want to perform maintenance on the TimesTen databases associated with this object.

```
% kubectl get ttc sample
NAME     STATE    ACTIVE     AGE
sample   Normal   sample-0   13m
```

3. Set the `.spec.ttspec.stopManaging` value. The value must be different than the current value for the TimesTenClassic object. When the Operator examines this value and notices it has changed since the last iteration, it will take appropriate action.

   Use the `kubectl edit` command to edit the TimesTenClassic object.

   - If there is a line for `.spec.ttspec.stopManaging` in the file, then modify its value. It must be different than the current value.
   - If there is no line for `.spec.ttspec.stopManaging` in the file, then add a line and specify a value.

   In this example, there is no `.spec.ttspec.stopManaging` line. This example adds the `.spec.ttspec.stopManaging` line and sets the value for `.spec.ttspec.stopManaging` to `April21Stop1` (represented in **bold**).

   Note: Not all output is shown:

```
% kubectl edit timestenclassic sample
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this
# file will be reopened with the relevant failures.
#
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
...
  name: sample
  namespace: mynamespace
...
repCreateStatement: |
  create active standby pair
```

```
        "{{tt-name}}" on "{{tt-node-0}}",
        "{{tt-name}}" on "{{tt-node-1}}"
      RETURN TWOSAFE
      store "{{tt-name}}" on "{{tt-node-0}}"
        PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999
      store "{{tt-name}}" on "{{tt-node-1}}"
        PORT {{tt-rep-port}} FAILTHRESHOLD 0 TIMEOUT 999
spec:
  ttspec:
    bothDownBehavior: Best
    dbConfigMap:
    - sample
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    storageClassName: oci
    storageSize: 250Gi
    stopManaging: April21Stop1
...
timestenclassic.timesten.oracle.com/sample edited
```

4. Use the `kubectl get` command to check the state of the `sample` TimesTenClassic object. Note that the `sample` TimesTenClassic object has transitioned to the `ManualInterventionRequired` state. This is the expected behavior after changing the `.spec.ttspec.stopManaging` value to a new value.

```
% kubectl get ttc sample
NAME      STATE                         ACTIVE     AGE
sample    ManualInterventionRequired    sample-0   15m
```

The `sample` TimesTenClassic object is in the `ManualInterventionRequired` state. The Operator has suspended the monitoring and the management of the `sample` TimesTenClassic object. It will take no further action on this TimesTenClassic object or its Pods. You can now perform manual operations on your TimesTen databases. When you have completed such operations and are ready for the Operator to resume management, proceed to Bringing up one database.

Locating the Operator

The Operator is configured in your Kubernetes cluster using a Deployment. Kubernetes automatically monitors the Operator and restarts it if it fails. The Operator runs in a Pod and the name of the Operator begins with `timesten-operator`, followed by arbitrary characters to make the name unique. If you specify multiple replicas when you deploy the Operator, there are multiple Pods. Only one Pod is active at a time. The remainder of the Pods wait for the active to fail, and if it does, then one of the Pods becomes active. Active standby pairs of TimesTen databases, provisioned by the Operator, continue to function if the Operator fails. When a new Operator is started by Kubernetes, it automatically monitors and manages all existing active standby pairs of databases.

Use the `kubectl get pods` command to display the Pods that are running the Operator. In this example, there is one Pod for the Operator. When you deployed the Operator, you specified the value of `1` for the `replicas` field. Therefore, Kubernetes created one Pod. See Deploy the TimesTen Operator.

```
% kubectl get pods
NAME                                        READY    STATUS     RESTARTS    AGE
timesten-operator-5d7dcc7948-8mnz4          1/1      Running    0           3m21s
```

Managing the TimesTen databases

The Operator strives to keep your active standby pair of databases running once they are deployed. Kubernetes manages the lifecycle of the Pods. It recreates the Pods if they fail. It also recreates the Pods on available Kubernetes cluster nodes, if the nodes on which the Pods are running fail. The Operator monitors TimesTen running in the Pods, and initiates the appropriate operations to keep the pair of databases operational. These operations are done automatically by the Operator, and should require minimal human intervention.

These sections discuss the manual operations you can perform:

- Manually invoke TimesTen utilities
- Modify TimesTen connection attributes
- Revert to manual control
- Delete an active standby pair of TimesTen databases

Manually invoke TimesTen utilities

You can use the `kubectl exec -it` command to manually invoke TimesTen utilities on your TimesTen instances. This command invokes shells in the Pods and enables you to control the running of TimesTen in the Pods.

TimesTen runs in the `tt` container, as the `timesten` user.

> **Note:**
>
> The Operator is still querying the status of the Pod, and the status of TimesTen within the Pod. If you invoke a command that disrupts the functioning of either the Pod or TimesTen, the Operator may act to try to fix what you did.

This example shows how to use the `kubectl exec -it` command to invoke the shell within the `sample-0` Pod that contains the TimesTen database. Then, you can run the `ttIsql` utility.

```
% kubectl exec -it sample-0 -c tt -- /bin/bash
% ttIsql sample

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
```

Modify TimesTen connection attributes

TimesTen uses connection attributes to define the attributes of a database. There are three types of connection attributes:

- Data store attributes: Define the characteristics of a database that can only be changed by destroying and recreating the database.
- First connection attributes: Define the characteristics of a database that can be changed by unloading and reloading the database into memory.
- General connection attributes: Control how applications access the database. These attributes persist for the duration of the connection.

For more information on TimesTen connection attributes, see List of attributes in the *Oracle TimesTen In-Memory Database Reference* and Connection attributes for Data Manager DSNs or Server DSNs in the *Oracle TimesTen In-Memory Database Operations Guide*.

In a Kubernetes environment:

- You can only modify data store attributes by deleting the TimesTenClassic object and the PersistentVolumeClaims associated with the TimesTenClassic object. Doing so results in the deletion of the TimesTen databases. See Delete an active standby pair of TimesTen databases and Cleanup for information on the deletion process.
- You can modify first connection and general connection attributes without deleting the TimesTenClassic object (which deletes the databases) and the PersistentVolumeClaims associated with the TimesTenClassic object. Note that there are TimesTen restrictions when modifying some of the first connection attributes. See List of attributes in the *Oracle TimesTen In-Memory Database Reference*.

To modify first or general connection attributes:

- You must first edit the `db.ini` file. Complete the procedure in the Manually edit the db.ini file section. This section must be completed first.

Then, take these steps:

- If you are modifying first connection attributes, follow the procedure in the Modifying first connection attributes section.
- If you are modifying general connection attributes, follow the procedure in the Modifying general connection attributes section.

Manually edit the db.ini file

Complete this section if you are modifying first or general connection attributes or both. This section must be completed before proceeding to the Modifying first connection attributes or the Modifying general connection attributes sections.

To modify first or general connection attribute requires a change in the `sys.odbc.ini` file.

If you have already created your active standby pair of TimesTen databases by creating a TimesTenClassic object, and you now want to change one or more first or general connection attributes in your `sys.odbc.ini` file, you must change the `db.ini` file.

The details as to how you should modify your `db.ini` file depends on the facility originally used to contain the `db.ini` file. (Possible facilities include ConfigMaps, Secrets, or init containers. See Populating the /ttconfig directory for details.)

In this example, the ConfigMap facility was originally used to contain the `db.ini` file and to populate the `/ttconfig` directory of the TimesTen containers. The example modifies the `sample` ConfigMap.

The steps are:

1. Use the `kubectl describe` command to review the contents of the `db.ini` file (represented in **bold**) located in the original `sample` ConfigMap.

   ```
   % kubectl describe configmap sample
   Name:         sample
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>

   Data
   ====
   adminUser:
   ----
   sampleuser/samplepw

   db.ini:
   ----
   PermSize=200
   DatabaseCharacterSet=AL32UTF8

   schema.sql:
   ----
   create sequence sampleuser.s;
   create table sampleuser.emp (id number not null primary key, name char (32));

   Events:  <none>
   ```

2. Use the `kubectl edit` command to modify the `db.ini` file in the original `sample` ConfigMap. Change the `PermSize` first connection attribute to `600` (represented in **bold**). Add the `TempSize` first connection attribute and set its value to `300` (represented in **bold**). Add the `ConnectionCharacterSet` general connection attribute and set its value to `AL32UTF8` (represented in **bold**).

   ```
   % kubectl edit configmap sample
   # Please edit the object below. Lines beginning with a '#' will be ignored,
   # and an empty file will abort the edit. If an error occurs while saving this
   # file will be reopened with the relevant failures.
   #
   apiVersion: v1
   data:
     adminUser: |
       sampleuser/samplepw
     db.ini: |
       PermSize=600
       TempSize=300
       ConnectionCharacterSet=AL32UTF8
       DatabaseCharacterSet=AL32UTF8
     schema.sql: |
       create sequence sampleuser.s;
   ```

```
    create table sampleuser.emp (id number not null primary key, name char
(32));
kind: ConfigMap
metadata:
  creationTimestamp: "2022-04-30T19:23:59Z"
  name: sample
  namespace: mynamespace
  resourceVersion: "71907255"
  selfLink: /api/v1/namespaces/mynamespace/configmaps/sample
 uid: 0171ff7f-f789-11ea-82ad-0a580aed0453
...
configmap/sample edited
```

3. Use the `kubectl describe` command to verify the changes to the `sample` ConfigMap. (The changes are represented in **bold**.)

```
% kubectl describe configmap sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
schema.sql:
----
create sequence sampleuser.s;
create table sampleuser.emp (id number not null primary key, name char (32));

adminUser:
----
sampleuser/samplepw

db.ini:
----
PermSize=600
TempSize=300
ConnectionCharacterSet=AL32UTF8
DatabaseCharacterSet=AL32UTF8

Events:  <none>
```

You have successfully changed the `sample` ConfigMap. If you are modifying first connection attributes, proceed to the Modifying first connection attributes section. If you are modifying only general connection attributes, proceed to the Modifying general connection attributes section.

Modifying first connection attributes

If you have not modified the `db.ini` file, proceed to the Manually edit the db.ini file section. You must now delete the standby Pod and then delete the active Pod. When you delete a Pod that contains a container running TimesTen, the Operator creates a new Pod to replace the deleted Pod. This new Pod contains a new `sys.odbc.ini` file which is created from the contents of the `db.ini` file located in the `/ttconfig` directory.

Perform these steps to delete the standby Pod.

1. Use the `kubectl get` command to determine which Pod is the standby Pod for the `sample` TimesTenClassic object. The active Pod is the Pod represented in the `ACTIVE` column. The standby Pod is the other Pod (not represented in the `ACTIVE`

column). Therefore, for the `sample` TimesTenClassic object, the active Pod is `sample-0`, (represented in **bold**) and the standby Pod is `sample-1`.

```
% kubectl get ttc sample
NAME      STATE    ACTIVE     AGE
sample    Normal   sample-0   47h
```

2. Delete the standby Pod (`sample-1`, in this example). This results in the Operator creating a new standby Pod to replace the deleted Pod. When the new standby Pod is created, it will use the newly modified `sample` ConfigMap. (You modified this ConfigMap in the Manually edit the db.ini file section.)

```
% kubectl delete pod sample-1
pod "sample-1" deleted
```

3. Use the `kubectl get` command to verify the standby Pod is up and running and the state is `Normal`.

   Note that the state is `StandbyDown` (represented in **bold**).

```
% kubectl get ttc sample
NAME      STATE         ACTIVE     AGE
sample    StandbyDown   sample-0   47h
```

   Wait a few minutes, then run the command again. Note that the state has changed to `Normal` (represented in **bold**).

```
% kubectl get ttc sample
NAME      STATE    ACTIVE     AGE
sample    Normal   sample-0   47h
```

4. Use the `kubectl exec -it` command to invoke the shell in the standby Pod (`sample-1`, in this example). Then, run the `ttIsql` utility to connect to the `sample` database. Note the new `PermSize` value of `600` and the new `TempSize` value of `300` in the connection output (represented in **bold**).

```
% kubectl exec -it sample-1 -c tt -- /bin/bash
% ttIsql sample
Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
AutoCreate=0;PermSize=600;TempSize=300;DDLReplicationLevel=3;
ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)
```

5. Fail over from the active Pod to the standby Pod. See "Fail over" for details of the fail over process. Before you begin this step, ensure you quiesce your applications and you use the `ttRepAdmin -wait` command to wait until replication is caught up, such that all transactions that were executed on the active database have been replicated to the standby database. Once the standby is caught up, fail over from the active database to the standby by deleting the active Pod. When you delete the active Pod, the Operator automatically detects the failure and promotes the standby database to be the active.

   Delete the active Pod (`sample-0`, in this example).

```
% kubectl delete pod sample-0
pod "sample-0" deleted
```

6. Wait a few minutes, then use the `kubectl get` command to verify the active Pod is now `sample-1` for the `sample` TimesTenClassic object and the state is `Normal` (represented in **bold**).

```
% kubectl get ttc sample
NAME      STATE     ACTIVE     AGE
sample    Normal    sample-1   47h
```

7. Use the `kubectl exec -it` command to invoke the shell in the active Pod (`sample-1`, in this example). Then, run the `ttIsql` utility to connect to the `sample` database. Note the new `PermSize` value of `600` and the new `TempSize` value of `300` in the connection output (represented in **bold**).

```
% kubectl exec -it sample-1 -c tt -- /bin/bash
Last login: Fri Apr 08 15:50:29 UTC 2022 on pts/0
[timesten@sample-1 ~]$ ttIsql sample

Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
AutoCreate=0;PermSize=600;TempSize=300;DDLReplicationLevel=3;
ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)
```

8. Use the `kubectl exec -it` command to invoke the shell in the standby Pod (`sample-0`, in this example). Then, run the `ttIsql` utility to connect to the `sample` database. Note the new `PermSize` value of `600` and the new `TempSize` value of `300` in the connection output (represented in **bold**).

```
% kubectl exec -it sample-0 -c tt -- /bin/bash
% ttIsql sample

Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
AutoCreate=0;PermSize=600;TempSize=300;DDLReplicationLevel=3;
ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)
```

You have successfully modified the `PermSize` and the `TempSize` first connection attributes.

Modifying general connection attributes

If you have not modified the `db.ini` file, proceed to the Manually edit the db.ini file section. You can either directly modify the `sys.odbc.ini` file for the active TimesTen database and the `sys.odbc.ini` file for the standby TimesTen database or you can follow the steps in the Modifying first connection attributes section. The first approach (modifying the `sys.odbc.ini` file directly) is less disruptive.

This section discusses the procedure for directly modifying the `sys.odbc.ini` files.

The `sys.odbc.ini` file is located in the TimesTen container of the Pod containing the active TimesTen database and in the TimesTen container of the Pod containing the standby TimesTen database. After you complete the modifications to the `sys.odbc.ini` files, subsequent applications can connect to the database using these general connection attributes.

This example illustrates how to edit the `sys.odbc.ini` files.

1. Use the `kubectl exec -it` command to invoke a shell in the active Pod. (In this example, `sample-0` is the active Pod.)

   ```
   % kubectl exec -it sample-0 -c tt -- /bin/bash
   Last login: Fri Apr 08 22:43:26 UTC 2022 on pts/8
   ```

2. Verify the current directory (`/tt/home/timesten`).

   ```
   % pwd
   /tt/home/timesten
   ```

3. Navigate to the directory where the `sys.odbc.ini` file is located. The `sys.odbc.ini` file is located in the `/tt/home/timesten/instances/instance1/conf` directory. Therefore, navigate to the `instances/instance1/conf` directory.

   ```
   % cd instances/instance1/conf
   ```

4. Edit the `sys.odbc.ini` file, adding, modifying, or deleting the general connection attributes for your DSN. (`sample`, in this example.) For a list of the general connection attributes, see List of attributes in the *Oracle TimesTen In-Memory Database Reference*.

   Note:

   Ensure that you only make modifications to the TimesTen general connection attributes. Data store attributes and first connection attributes cannot be modified by directly editing the `sys.odbc.ini` file.

   This example modifies the `sample` DSN, adding the `ConnectionCharacterSet` general connection attribute and setting its value equal to `AL32UTF8` (represented in **bold**).

   ```
   vi sys.odbc.ini

   [ODBC Data Sources]
   sample=TimesTen 22.1 Driver
   tt=TimesTen 22.1 Driver

   [sample]
   Datastore=/tt/home/timesten/datastore/sample
   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ConnectionCharacterSet=AL32UTF8
   DDLReplicationLevel=3
   AutoCreate=0
   ForceDisconnectEnabled=1
   ...
   ```

5. Use the `ttIsql` utility to connect to the `sample` database and verify the value of the `ConnectionCharacterSet` attribute is `AL32UTF8` (represented in **bold**).

   ```
   % ttIsql sample

   Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=sample";
   Connection successful:
   DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
   AutoCreate=0;PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=1;
   (Default setting AutoCommit=1)
   ```

You have successfully modified the `sys.odbc.ini` file located in the TimesTen container of the active Pod (in this example, `sample-0`). Use the same procedure to modify the `sys.odbc.ini` file located in the TimesTen container of the standby Pod (in this example, `sample-1`).

For example:

1. Use the `kubectl exec -it` command to invoke a shell in the standby Pod (`sample-1`, in this example).

   ```
   % kubectl exec -it sample-1 -c tt -- /bin/bash
   Last login: Fri Apr 08 23:08:08 UTC 2022 on pts/0
   ```

2. Verify the current directory (`/tt/home/timesten`).

   ```
   % pwd
   /tt/home/timesten
   ```

3. Navigate to the directory where the `sys.odbc.ini` file is located. The `sys.odbc.ini` file is located in the `/tt/home/timesten/instances/instance1/conf` directory. Therefore, navigate to the `instances/instance1/conf` directory.

   ```
   % cd instances/instance1/conf
   ```

4. Edit the `sys.odbc.ini` file, adding, modifying, or deleting the same general connection attributes that you modified for the active database. Therefore, this example modifies the `sample` DSN, adding the `ConnectionCharacterSet` general connection attribute and setting its value equal to `AL32UTF8` (represented in **bold**).

   ```
   vi sys.odbc.ini

   [ODBC Data Sources]
   sample=TimesTen 22.1 Driver
   tt=TimesTen 22.1 Driver

   [sample]
   Datastore=/tt/home/timesten/datastore/sample
   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ConnectionCharacterSet=AL32UTF8
   DDLReplicationLevel=3
   AutoCreate=0
   ForceDisconnectEnabled=1
   ...
   ```

5. Use the `ttIsql` utility to connect to the `sample` database and verify the value of the `ConnectionCharacterSet` attribute is `AL32UTF8` (represented in **bold**).

   ```
   % ttIsql sample

   Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=sample";
   Connection successful:
   DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
   AutoCreate=0;PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=1;
   (Default setting AutoCommit=1)
   ```

You have successfully modified the `sys.odbc.ini` file located in the TimesTen container of the active Pod (`sample-0`) and the `sys.odbc.ini` file located in the TimesTen container of the standby Pod (`sample-1`). The `ConnectionCharacterSet` general connection attribute has also been modified.

Revert to manual control

If you want to manually operate your active standby pair, you can delete the `timesten-operator` Deployment. The Operator stops, and does not restart. This affects all of the TimesTenClassic objects that are running in your Kubernetes cluster. If you do not want the Operator to stop managing all of the TimesTenClassic objects, you can suspend the management of individual TimesTenClassic objects. See "Suspending the management of a TimesTenClassic object" for information.

The TimesTenClassic object, representing the active standby pair of TimesTen databases, remains in Kubernetes, as do the other Kubernetes objects associated with them. You can use the `kubectl exec -it` command to invoke shells in the Pods, and then you can control Timesten that is running in those Pods.

If one or both Pods in your active standby pair fails, Kubernetes creates new ones to replace them. This is due to the StatefulSet object that the Operator had previously created in Kubernetes. However, since the Operator is not running the new Pods, it cannot automatically start TimesTen. In this case, your active standby pair cannot be configured or started. You are responsible for the operation of TimesTen in the Pods.

If you want the Operator to take control again, you must redeploy the Operator. Once the Operator is redeployed, the Operator automatically identifies the TimesTenClassic objects in your Kubernetes cluster, and will attempt to manage them again.

This example shows you how to manually control TimesTen.

1. Verify the Operator and the TimesTen databases are running.

   ```
   % kubectl get pods
   NAME                                    READY   STATUS    RESTARTS   AGE
   sample-0                                2/2     Running   0          18h
   sample-1                                2/2     Running   0          18h
   timesten-operator-5d7dcc7948-pzj58      1/1     Running   0          18h
   ```

2. Navigate to the `/deploy` directory where the `operator.yaml` resides. (*kube_files*/`deploy`, in this example.)

   ```
   % cd kube_files/deploy
   ```

3. Use the `kubectl delete` command to delete the Operator. The Operator is stopped and no longer deployed.

   ```
   % kubectl delete -f operator.yaml
   deployment.apps "timesten-operator" deleted
   ```

4. Verify the Operator is no longer running, but the TimesTen databases are.

   ```
   % kubectl get pods
   NAME        READY   STATUS    RESTARTS   AGE
   sample-0    2/2     Running   0          19h
   sample-1    2/2     Running   0          19h
   ```

5. Use the `kubectl exec -it` command to invoke the shell in the Pod that runs TimesTen.

   ```
   % kubectl exec -it sample-0 -c tt -- /bin/bash
   Last login: Fri Apr  8 14:30:45 UTC 2022 on pts/0
   ```

6. Run the `ttStatus` utility.

   ```
   % ttStatus
   TimesTen status report as of Fri Apr  8 14:36:31 2022

   Daemon pid 183 port 6624 instance instance1
   TimesTen server pid 190 started on port 6625
   ------------------------------------------------------------------------
   ------------------------------------------------------------------------
   Data store /tt/home/timesten/datastore/sample
   Daemon pid 183 port 6624 instance instance1
   TimesTen server pid 190 started on port 6625
   There are 20 connections to the data store
   Shared Memory KEY 0x02200bbc ID 32769
   PL/SQL Memory Key 0x03200bbc ID 65538 Address 0x5000000000
   Type            PID     Context            Connection Name             ConnID
   Replication     263     0x00007f99fc0008c0 LOGFORCE:140299698493184      2029
   Replication     263     0x00007f9a040008c0 XLA_PARENT:140300350273280    2031
   Replication     263     0x00007f9a080008c0 REPLISTENER:140300347123456   2030
   Replication     263     0x00007f9a080acd60 RECEIVER:140299429472000      2028
   Replication     263     0x00007f9a0c0008c0 FAILOVER:140300353423104      2032
   Replication     263     0x00007f9a2c0009b0 TRANSMITTER(M):140299695343360
   2034
   Replication     263     0x00007f9a300008c0 REPHOLD:140300356572928
   2033
   Subdaemon       187     0x00000000023365b0 Manager
   2047
   Subdaemon       187     0x00000000023b57f0 Rollback
   2046
   Subdaemon       187     0x0000000002432cf0 Log Marker
   2041
   Subdaemon       187     0x000000000244fc00 Garbage Collector
   2035
   Subdaemon       187     0x00007f90c80008c0 Aging
   2038
   Subdaemon       187     0x00007f90d00008c0 Deadlock Detector
   2044
   Subdaemon       187     0x00007f90d001d7d0 HistGC
   2039
   Subdaemon       187     0x00007f90d40008c0 Checkpoint
   2042
   Subdaemon       187     0x00007f90d401d7d0 AsyncMV
   2036
   Subdaemon       187     0x00007f90d80008c0 Monitor
   ```

```
2043
Subdaemon       187     0x00007f90f808b360 IndexGC                         2037
Subdaemon       187     0x00007f90fc0008c0 Flusher                         2045
Subdaemon       187     0x00007f910004efd0 XactId Rollback                 2040
Open for user connections
RAM residence policy: Always
Replication policy  : Manual
Replication agent is running.
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group timesten
End of report
```

7. Run the `ttIsql` utility to connect to the `sample` database and perform various operations.

```
% ttIsql sample

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
Command> describe sampleuser.emp;

Table SAMPLEUSER.EMP:
  Columns:
   *ID                             NUMBER NOT NULL
    NAME                           CHAR (32)

1 table found.
(primary key columns are indicated with *)

Command> INSERT INTO sampleuser.emp VALUES (1,'This is a test.');
1 row inserted.
Command> SELECT * FROM sampleuser.emp;
< 1, This is a test.                  >
1 row found.
```

Delete an active standby pair of TimesTen databases

If you delete the TimesTenClassic object that represents the active standby pair of TimesTen databases, Kubernetes automatically deletes all the Kubernetes objects and the resources it is using. The StatefulSet, the Service, and the Pods, that are associated with the pair are all deleted from Kubernetes. However, the PersistentVolumeClaims (that contain the TimesTen databases) are not deleted. You must manually delete the PersistentVolumeClaims (PVCs) after you delete the TimesTenClassic object. After you manually delete the PVCs, the PersistentVolumes, holding the databases, are recycled by Kubernetes. (You may be able to control this using the Kubernetes volume retention policy, but this is not controlled by the Operator.)

As an example, use the `kubectl delete` command to delete the PVCs for the `sample` databases.

```
% kubectl delete pvc tt-persistent-sample-0
persistentvolumeclaim "tt-persistent-sample-0" deleted
% kubectl delete pvc tt-persistent-sample-1
persistentvolumeclaim "tt-persistent-sample-1" deleted
```

9

Managing TimesTen Scaleout

This chapter discusses how the TimesTen Operator manages and repairs TimesTen Scaleout.

Topics:

Overview

TimesTen Scaleout delivers high performance, fault tolerance, and scalability within a highly available in-memory database that provides persistence and recovery. Since a database is distributed across multiple hosts, some components of the database may fail while others continue to operate.

TimesTen Scaleout supports error and failure detection with automatic recovery for many error and failure situations in order to maintain a continuous operation for all applications.

The TimesTen Operator implements best practices for how to handle failures for TimesTen Scaleout. For more information about how TimesTen Scaleout handles failures, see Recovering From Failure in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*

In particular, the Operator detects and handles the following failure cases:

- If a TimesTen instance or element fails, the Operator restarts it.
- If an entire replica set fails and if all elements in the replica set reach the `waiting for seed` state, the Operator unloads and reloads the database to resolve it (by default). For

details about how TimesTen Scaleout recovers from a down replica set, see Recovering From a Down Replica Set in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

- If all data instances fail, the Operator detects and reports the failure.

The Operator communicates to the TimesTen agent running in the `tt` container in each Pod running TimesTen. The agent determines information about the state of TimesTen running in the container and sends that information back to the Operator. The Operator analyzes this information and determines the health and state of TimesTen. This information is summarized in well-defined states. The Operator uses state machines to determine the appropriate set of commands to be executed to detect failures and, if possible, repair TimesTen. These states are discussed later in the chapter.

Let's take a deeper look at how the Operator detects and repairs TimesTen Scaleout. Specifically, let's look at how the Operator handles single data instance failure, management instance failure, entire replica set failure, and total database failure.

About single data instance failure

TimesTen Scaleout is intended to be resilient to single data instance failures. In your Kubernetes environment, if a data instance fails, the TimesTen Operator starts it back up. Once it is restarted, TimesTen Scaleout reloads the element of the database. See Recovering When a Data Instance Is Down in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

About management instance failure

The TimesTen Operator supports one management instance per TimesTenScaleout object. If the management instance is down, the Operator ignores the state of the data instances until the management instance is back up. Without a management instance, the grid and TimesTen database continue to function. However, should a data instance fail without a working management instance, it is not possible to repair the data instance.

About waiting for seed

In TimesTen Scaleout, there could be a situation where an entire replica set fails simulataneously. If all elements in a replica set fail at the same time, it may not be possible to reload any of the elements without unloading the entire database and reloading it. In addition, a reload may cause TimesTen Scaleout to discard transactions against other replica sets that were committed prior to the unload operation.

For more information about replica set failure in TimesTen Scaleout, see Recovering From a Down Replica Set in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

If this case occurs in your Kubernetes environment, the default behavior is for the TimesTen Operator to detect this situation and forcibly unload and reload the database when the situation occurs. This operation is triggered when the Operator notices that all elements in a replica set are in the `waiting for seed` state.

You have the option of controlling the behavior of the Operator when all elements in a replica set are in the `waiting for seed` state. You do this by using a TimesTenScaleout object's `.spec.ttspec.replicaSetRecovery` datum, and setting a particular value for the datum. Accepted values are the following:

- `Restart`: The Operator forcibly unloads and reloads the database when a total replica set failure occurs. This is the default.
- `Manual`: The Operator changes the state of the TimesTenScaleout object to `ManualInterventionRequired` when a total replica set failure occurs. The Operator takes no further action to repair the grid. You must repair it. See About the ManualInterventionRequired state and Set reexamine for details.

For more information on the `.spec.ttspec.replicaSetRecovery` datum for a TimesTenScaleout object, see TimesTenScaleoutSpecSpec.

About failure of all data instances

If all elements in a database fail simultaneously, there must be caution taken in reloading the database. TimesTen Scaleout does not automatically reload a database if all data instances fail. You must perform this reload. It is essential that all possible data instances are up before a reload is attempted.

For more information about failure of all data instances in TimesTen Scaleout, see Database Recovery in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

If this case occurs in a Kubernetes environment, the TimesTen Operator does not attempt to reload the database after all data instances fail. However, the Operator does detect this case and changes the state of a TimesTenScaleout object to `ManualInterventionRequired`. See About the ManualInterventionRequired state and Set reexamine.

About High Level states

The TimesTen Operator maintains a High Level state for TimesTenScaleout objects.

These High Level states are as follows:

- DatabaseDown
- DatabaseImpeded
- DatabasePartial
- DatabaseRestarting
- DatabaseRestartRequired
- Failed
- Initializing
- ManualInterventionRequired
- Normal
- Reexamine
- Unmanaged

DatabaseDown

The database is unusable. The Operator attempts to fix it. If it cannot be fixed, the Operator moves the object to the ManualInterventionRequired state.

DatabaseImpeded

The database within the grid is fully operational, but one or more elements are not functional. All data in the database is available and all SQL is accepted.

DatabasePartial

The database is up, but some data is not available. One or more replica sets are not available.

DatabaseRestarting

The database is in the process of being forcibly unloaded and reloaded after a `DatabaseRestartRequired` condition.

DatabaseRestartRequired

While the database is up (at least partially), it must be stopped and restarted (unloaded and reloaded) in order to restore functionality. This can occur when all elements in a replica set fail simultaneously and all elements are unloadable due to a `waiting for seed` condition. When this occurs, the database must be unloaded and reloaded. During this time, committed transactions may be lost. See Database Recovery and Recovering From a Down Replica Set in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Failed

If a problem occurs while `Initializing` a TimesTenScaleout object, the object transitions to the `Failed` state. Once in this state, the Operator does not attempt to repair the object. You must delete it. Use the `kubectl describe` command to examine the Operator logs to determine the cause of the problem and then recreate the object.

Initializing

After you create a TimesTenScaleout object in your Kubernetes cluster, the Operator creates the StatefulSets and Services that are required to deploy a TimesTenScaleout grid and database. The Operator assigns a High Level state of `Initializing` to the TimesTenScaleout object.

ManualInterventionRequired

If the Operator encounters a problem that it does not know how to fix, the Operator places a TimesTenScaleout object into this state. The Operator takes no further action to fix the object. You can set the object's `.spec.ttspec.reexamine` datum to cause the

Operator to re-engage with the object. For information about `.spec.ttspec.reexamine`, see Set reexamine.

Normal

The grid and database are up and operating as they should.

Reexamine

If a TimesTenScaleout object is in the `ManualInterventionRequired` state, and you modify the object's `.spec.ttspec.reexamine` datum, the TimesTen Operator moves the object into the `Reexamine` state. The Operator examines the state of grid and database. If healthy, the Operator returns the object to the `Normal` state. Otherwise, the object re-enters the `ManualInterventionRequired` state.

Unmanaged

The grid does not have a functional management instance. Until the management instance is fixed, the grid cannot be further monitored, managed or controlled. The Operator will attempt to fix the management instance.

About management states

The Operator uses the TimesTen `ttGridAdmin mgmtExamine` utility to determine the health of the management instance. The Operator also synthesizes a management state that describes the status of the management instance in a single value.

For information about the TimesTen `ttGridAdmin mgmtExamine` utility, see Examine management instances (mgmtExamine) in the *Oracle TimesTen In-Memory Database Reference*.

For example, you can use the `kubectl get tts` command to observe the state of a TimesTenScaleout object.

```
kubectl get tts samplescaleout
NAME             OVERALL    MGMT     CREATE    LOAD              OPEN   AGE
samplescaleout   Normal     Normal   created   loaded-complete   open   10m
```

Note that the management state is **Normal**. There are additional examples in the chapter where you have the opportunity to observe the management state.

Here are the management states:

- ActiveAgentUp
- ActiveDaemonUp
- ActiveDown
- Error
- Normal
- Unknown

ActiveAgentUp

The TimesTen agent in the container that should be running the management instance is up, but the management instance has not yet been started.

ActiveDaemonUp

The management instance has been started, but is not yet functional.

ActiveDown

The management instance is down.

Error

There is an unexpected error with the management instance.

Normal

The management instance is functioning normally.

Unknown

The state of the management instance cannot be determined.

About database and element states

TimesTen Scaleout defines a set of overall database and element status values to detemine the status of a database or element. The TimesTen Operator uses these status values to assess the state of the database represented by a TimesTenScaleout object.

In TimesTen Scaleout, the overall database status is encoded in three strings:

- How created is the database?
- How loaded is the database?
- How open is the database?

This triplet of strings is returned as part of the output of the TimesTen `ttGridAdmin dbStatus` utility.

For information about the database states, see Display the Status of the Database and All Elements in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*. For information about the `ttGridAdmin dbstatus` utility, see Monitor the status of a database (dbStatus) in the *Oracle TimesTen In-Memory Database Reference*.

The TimesTen Operator uses TimesTen Scaleout database states to report the state of the database for a TimesTenScaleout object.

For example, this code snippet uses the `kubectl get` command to return the status for a deployed TimesTenScaleout object.

```
kubectl get tts samplescaleout
NAME              OVERALL                       MGMT      CREATE
LOAD                  OPEN    AGE
samplescaleout    Normal                        Normal    created    loaded-
complete       open    2d
```

Note the following:

- The High Level state of the TimesTenScaleout object is `Normal` (as indicated by the `OVERALL` field).
- The management instance state is `Normal` (as indicated by the `MGMT` field).
- The database `creation` state is `created` (as indicated by the `CREATE` field).
- The database `loaded` state is `loaded-complete` (as indicated by the `LOAD` field).
- The database `open` state is `open` (as indicated by the `OPEN` field).

TimesTen Scaleout also keeps a state for each element in a database. See Troubleshooting Based on Element Status in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for details on element states.

The Operator does not monitor these element states. However, if all the elements in a replica set are in the `waiting for seed` state, the Operator checks the value of a TimesTenScaleout object's `.spec.ttspec.replicaSetRecovery` datum:

- If the value is `Restart`, the `DatabaseRestartRequired` High Level state is triggered.
- If the value is `Manual`, the Operator moves the TimesTenScaleout object to the `ManualInterventionRequired` state.

About the ManualInterventionRequired state

If the TimesTen Operator determines it cannot repair a TimesTenScaleout object, the Operator changes the High Level state of the object to `ManualInterventionRequired`. The Operator does not further manage an object in this state. In addition, the Operator makes no attempt to determine its state nor to repair it.

In TimesTen Scaleout, there are several troubleshooting scenarios that you can review to identify and possibly fix the problem. For more information about the troubleshooting scenarios, see Recovering From Failure in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

If you identify and fix the problem, you can cause the Operator to resume management of a TimesTenScaleout object. You do this by specifying a TimesTenScaleout object's `spec.ttspec.reexamine` datum. When this datum is specified, the Operator moves the object to the `Reexamine` state. For information about the `Reexamine` state, see Set reexamine.

About suspending management

There may be a situation in which you want to manually perform maintenance operations. In such a situation, you do not want the Operator to interfere and attempt to perform repair or recovery operations on your grid and database.

One alternative is to stop the Operator (by deleting the `timesten-operator` Deployment). This action prevents the Operator from interfering or performing repair/ recovery operations. However, if you have more than one TimesTenScaleout object deployed in your Kubernetes environment and you delete the Operator, this interferes with the management of all the TimesTenScaleout objects, when perhaps only one of them needs manual intervention.

Another approach is to ask the Operator to take no action for one TimesTenScaleout object. To do this, specify a TimesTenScaleout object's `.spec.ttspec.stopManaging` datum. The Operator examines the value of `.spec.ttspec.stopManaging` and if it has changed since the last time the Operator examined it, the Operator changes the state of the TimesTenScaleout object to `ManualInterventionRequired`. This causes the Operator to no longer examine the status of the grid and database. Nor does the Operator examine the Pods, the containers, and the instances associated with this particular TimesTenScaleout object. For an example showing how to set the `.spec.ttspec.stopManaging` datum for a TimesTenScaleout object, see Suspend management.

To cause the Operator to resume management of the TimesTenScaleout object, change the value of the object's `.spec.ttspec.reexamine` datum. See Set reexamine for details.

See TimesTenScaleoutSpecSpec for information about the TimesTenScaleout object definition.

Simulate single data instance failure

Let's simulate a single data instance failure and observe how a TimesTenScaleout object transitions through various state changes.

Note:

This example is for demonstration purposes only. Do not attempt this example in a production environment.

In the example, there is a deployed TimesTenScaleout object that is functioning properly.

```
kubectl get tts samplescaleout
NAME            OVERALL    MGMT      CREATE    LOAD               OPEN
AGE
samplescaleout  Normal     Normal    created   loaded-complete    open
10m
```

Note the High Level state is `Normal`, the management state is `Normal`, and the database state is `created,loaded-complete,open`.

To simulate a single data instance failure, let's delete a Pod that contains a data instance. Here are the Pods:

```
kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
```

```
samplescaleout-data-1-0             2/2     Running   0          11m
samplescaleout-data-1-1             2/2     Running   0          11m
samplescaleout-data-1-2             2/2     Running   0          11m
samplescaleout-data-2-0             2/2     Running   0          11m
samplescaleout-data-2-1             2/2     Running   0          11m
samplescaleout-data-2-2             2/2     Running   0          11m
samplescaleout-mgmt-0               2/2     Running   0          11m
samplescaleout-zk-0                 1/1     Running   0          11m
samplescaleout-zk-1                 1/1     Running   0          10m
samplescaleout-zk-2                 1/1     Running   0          9m35s
timesten-operator-7677964df9-sp2zp  1/1     Running   0          7d3h
```

Let's delete the `samplescaleout-data-1-0` Pod and observe the behavior.

1. Delete the Pod.

   ```
   kubectl delete pod samplescaleout-data-1-0
   pod "samplescaleout-data-1-0" deleted
   ```

2. Use the `kubectl get` command to observe state transitions.

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL           MGMT     CREATE     LOAD
   OPEN   AGE
   samplescaleout   DatabaseImpeded   Normal   created    loaded-functional
   open   16m
   ```

   The High Level state is `DatabaseImpeded`, indicating that the database within the grid is fully operational, but one or more elements is not functional. The database `loaded` state is `loaded-functional`, indicating loading is in progress and at least one element from each replica set is loaded.

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL   MGMT     CREATE    LOAD              OPEN   AGE
   samplescaleout   Normal    Normal   created   loaded-complete   open   18m
   ```

   The object transitioned to the `Normal` High Level state, indicating the grid and database are functioning normally. The database state is `loaded-complete`, indicating the element loaded successfully.

Even though there was a single data instance failure, TimesTen Scaleout fully recovered. There was no manual intervention required.

Simulate management instance failure

Let's simulate a management instance failure. Let's observe how a TimesTenScaleout object transitions through various state changes.

Note:

This example is for demonstration purposes only. Do not attempt this example in a production environment.

In the example, there is a deployed TimesTenScaleout object that is functioning properly.

```
kubectl get tts samplescaleout
NAME             OVERALL   MGMT     CREATE    LOAD               OPEN
AGE
samplescaleout   Normal    Normal   created   loaded-complete    open
68m
```

Note the High Level state is `Normal`, the management state is `Normal`, and the database state is `created,loaded-complete,open`.

To simulate a management instance failure, let's delete the Pod that contains the management instance. Here are the Pods:

```
kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
samplescaleout-data-1-0              2/2     Running   0          57m
samplescaleout-data-1-1              2/2     Running   0          73m
samplescaleout-data-1-2              2/2     Running   0          73m
samplescaleout-data-2-0              2/2     Running   0          73m
samplescaleout-data-2-1              2/2     Running   0          73m
samplescaleout-data-2-2              2/2     Running   0          73m
samplescaleout-mgmt-0                2/2     Running   0          73m
samplescaleout-zk-0                  1/1     Running   0          73m
samplescaleout-zk-1                  1/1     Running   0          72m
samplescaleout-zk-2                  1/1     Running   0          71m
timesten-operator-7677964df9-sp2zp   1/1     Running   0          7d4h
```

Let's delete the `samplescaleout-mgmt-0` Pod and observe the behavior.

1. Delete the Pod.

   ```
   kubectl delete pod samplescaleout-mgmt-0
   pod "samplescaleout-mgmt-0" deleted
   ```

2. Use the `kubectl get` command to observe state transitions.

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL     MGMT         CREATE
   LOAD              OPEN    AGE
   samplescaleout   Unmanaged   ActiveDown   created   loaded-
   complete   open    79m
   ```

   The High Level state is `Unmanaged`, indicating that the grid has no functional management instance. As a result, the grid cannot be further managed, monitored, or controlled. The management instance state is `ActiveDown`, indicating the management instance is down. Note that since the management instance is down, the Operator ignores the state of the data instances until the management instance is back up.

```
kubectl get tts samplescaleout
NAME             OVERALL     MGMT              CREATE     LOAD
OPEN    AGE
samplescaleout   Unmanaged   ActiveDaemonUp    created    loaded-complete
open    81m
```

The object remains in the `Unmanaged` High Level state. The management state transitions to `ActiveDaemonUp`, indicating the management instance has been started, but is not yet functional.

```
kubectl get tts samplescaleout
NAME             OVERALL   MGMT     CREATE    LOAD              OPEN   AGE
samplescaleout   Normal    Normal   created   loaded-complete   open   81m
```

The object transitioned to the `Normal` High Level state, indicating the grid and database are functioning normally. The management state is `Normal`, indicating that management instance is functioning normally.

Even though there was a management instance failure, TimesTen Scaleout fully recovered. There was no manual intervention required.

Simulate replica set failure with restart

Let's simulate a replica set failure. In this example, a TimesTenScaleout object's `.spec.ttspec.replicaSetRecovery` datum has not been specified. The default of `Restart` is assumed, indicating that the TimesTen Operator forcibly unloads and reloads the database when a total replica set failure occurs.

Let's observe how a TimesTenScaleout object transitions through various state changes.

Note:

This example is for demonstration purposes only. Do not attempt this example in a production environment.

In the example, there is a deployed TimesTenScaleout object that is functioning properly.

```
kubectl get tts samplescaleout
NAME             OVERALL   MGMT     CREATE    LOAD              OPEN   AGE
samplescaleout   Normal    Normal   created   loaded-complete   open   99m
```

Note the High Level state is `Normal`, the management state is `Normal`, and the database state is `created,loaded-complete,open`.

In this example, there are three replica sets.

```
kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
samplescaleout-data-1-0               2/2     Running   0          11m
samplescaleout-data-1-1               2/2     Running   0          11m
samplescaleout-data-1-2               2/2     Running   0          11m
samplescaleout-data-2-0               2/2     Running   0          11m
samplescaleout-data-2-1               2/2     Running   0          11m
samplescaleout-data-2-2               2/2     Running   0          11m
samplescaleout-mgmt-0                 2/2     Running   0          11m
samplescaleout-zk-0                   1/1     Running   0          11m
samplescaleout-zk-1                   1/1     Running   0          10m
samplescaleout-zk-2                   1/1     Running   0          9m35s
timesten-operator-7677964df9-sp2zp    1/1     Running   0          7d3h
```

Let's delete the `samplescaleout-data-1-0` and `samplescaleout-data-2-0` Pods that belong to one of the replica sets.

1. Delete the Pods.

   ```
   kubectl delete pod samplescaleout-data-1-0;kubectl delete pod
   samplescaleout-data-2-0
   pod "samplescaleout-data-1-0" deleted
   pod "samplescaleout-data-2-0" deleted
   ```

2. Use the `kubectl get` command to observe state transitions.

   ```
   kubectl get tts samplescaleout
   NAME              OVERALL            MGMT     CREATE
   LOAD                 OPEN    AGE
   samplescaleout    DatabasePartial    Normal   created   loaded-
   incomplete   open    111m
   ```

   The High Level state is `DatabasePartial` indicating that the database is up, but some data is not available. One or more replica sets have failed completely. The database `loaded` state is `loaded-incomplete`, indicating that at least one replica set has no elements that finished loading successfully.

   ```
   kubectl get tts samplescaleout
   NAME              OVERALL                    MGMT     CREATE
   LOAD                   OPEN    AGE
   samplescaleout    DatabaseRestartRequired    Normal   created
   loading-incomplete    open    112m
   ```

   The object transitions to the `DatabaseRestartRequired` High Level state. This state indicates that while the database is up (at least partially), the database must be stopped and restarted (unloaded and reloaded) in order to restore functionality. This can occur when all elements in a replica set fail simultaneously and such elements are unloadable due to a `waiting for seed` condition. When this happens the database must be unloaded and reloaded. At that time, committed transactions may be lost.

```
kubectl get tts samplescaleout
NAME              OVERALL              MGMT     CREATE
LOAD                  OPEN   AGE
samplescaleout    DatabaseRestarting   Normal   created   loading-
incomplete   open   114m
```

The object transitions to the `DatabaseRestarting` High Level state. This state indicates that the database is being forcibly unloaded and reloaded after a `DatabaseRestartRequired` condition.

```
kubectl get tts samplescaleout
NAME              OVERALL   MGMT     CREATE    LOAD              OPEN
AGE
samplescaleout    Normal    Normal   created   loaded-complete   closed
114m
```

The object transitions to the `Normal` High Level state, indicating that the grid and database are functioning normally. The database `loaded` state changed to `loaded-complete`, indicating every element was loaded successfully. The database `open` state is `closed`, indicating the database is closed for connections.

```
kubectl get tts samplescaleout
NAME              OVERALL   MGMT     CREATE    LOAD              OPEN   AGE
samplescaleout    Normal    Normal   created   loaded-complete   open
114m
```

The object remains in the `Normal` High Level state. The database `open` state changed to `open`, indicating the database is now open for connections.

Even though there was a total replica set failure, TimesTen Scaleout forcibly unloaded and reloaded the database. The Operator returned the grid and database to a Normal state, indicating both are functioning normally. There was no manual intervention required.

Simulate replica set failure with manual

Let's simulate a total replica set failure where a TimesTenScaleout object's `.spec.ttspec.replicaSetRecovery` datum has a value of `Manual`.

Note:

This example is for demonstration purposes only. Do not attempt this example in a production environment.

Let's take a look at the TimesTenScaleout object definition.

```
cat samplescaleout2.yaml
apiVersion: timesten.oracle.com/v1
```

```
kind: TimesTenScaleout
metadata:
  name: samplescaleout2
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    replicaSetRecovery: Manual
    dbConfigMap:
    - samplescaleout2
    k: 2
    nReplicaSets: 3
    nMgmt: 1
    nZookeeper: 3
```

Note the `.spec.ttspec.replicaSetRecovery` datum for the object has a value of `Manual`, indicating that the Operator will set this TimesTenScaleout object's High Level state to `ManualInterventionRequired` when a total replica set failure occurs.

Let's simulate a total replica set failure with this object. Before we begin, let's do a quick check of the state of the object.

```
kubectl get tts samplescaleout2
NAME                 OVERALL    MGMT      CREATE     LOAD
OPEN    AGE
samplescaleout2      Normal     Normal    created    loaded-complete
open    11m
```

Note the High Level state is `Normal`, the management state is `Normal`, and the database state is `created,loaded-complete,open`.

In this example, there are three replica sets.

```
kubectl get pods
NAME                                  READY    STATUS     RESTARTS    AGE
samplescaleout2-data-1-0              2/2      Running    0           12m
samplescaleout2-data-1-1              2/2      Running    0           12m
samplescaleout2-data-1-2              2/2      Running    0           12m
samplescaleout2-data-2-0              2/2      Running    0           12m
samplescaleout2-data-2-1              2/2      Running    0           12m
samplescaleout2-data-2-2              2/2      Running    0           12m
samplescaleout2-mgmt-0                2/2      Running    0           13m
samplescaleout2-zk-0                  1/1      Running    0           13m
samplescaleout2-zk-1                  1/1      Running    0           12m
samplescaleout2-zk-2                  1/1      Running    0           11m
timesten-operator-7677964df9-sp2zp    1/1      Running    0           7d6h
```

Let's delete the `samplescaleout2-data-1-0` and `samplescaleout2-data-2-0` Pods that belong to one of the replica sets.

1. Delete the Pods.

   ```
   kubectl delete pod samplescaleout2-data-1-0;kubectl delete pod
   samplescaleout2-data-2-0
   pod "samplescaleout2-data-1-0" deleted
   pod "samplescaleout2-data-2-0" deleted
   ```

2. Use the `kubectl get` command to observe state transitions.

   ```
   kubectl get tts samplescaleout2
   NAME              OVERALL           MGMT     CREATE
   LOAD                OPEN    AGE
   samplescaleout2   DatabasePartial   Normal   created   loaded-
   incomplete   open    18m
   ```

   The High Level state is `DatabasePartial` indicating that the database is up, but some data is not available. One or more replica sets have failed completely. The database `loaded` state is `loaded-incomplete`, indicating that at least one replica set has no elements that finished loading successfully.

   ```
   kubectl get tts samplescaleout2
   NAME              OVERALL                       MGMT     CREATE
   LOAD                OPEN    AGE
   samplescaleout2   ManualInterventionRequired    Normal   created   loading-
   incomplete   open    20m
   ```

   The object transitions to the `ManualInterventionRequired` High Level state. The Operator takes no further action to fix the object. The database `loaded` state remains `loading-incomplete`. Recall that you can set the `.spec.ttspec.reexamine` datum to cause the Operator to re-engage with the object. See Set reexamine for details.

There was a total replica set failure. Because the TimesTen Scaleout object's `.spec.ttspec.replicaSetRecovery` datum had a value of `Manual`, the Operator set the object to the `ManualInterventionRequired` state. Review the information in Recovering From a Down Replica Set in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for details about how to fix this failure. Next, review Set reexamine in this book to give control back to the Operator.

Suspend management

Let's walk through an example that shows you how to suspend management of a TimesTenScaleout object.

In the example, there is a deployed TimesTenScaleout object that is functioning properly.

```
kubectl get tts samplescaleout
NAME             OVERALL   MGMT     CREATE    LOAD                OPEN   AGE
samplescaleout   Normal    Normal   created   loaded-complete     open   3h33m
```

Note the High Level state is `Normal`, the management state is `Normal`, and the database state is `created,loaded-complete,open`.

1. Use the `kubectl edit` command to edit the TimesTenScaleout object, making the following changes:
   - If there is a line for `.spec.ttspec.stopManaging` in the file, then modify its value. It must be different than the current value.
   - If there is no line for `.spec.ttspec.stopManaging` in the file, then add a line and specify a value.

   In this example, there is no `.spec.ttspec.stopManaging` line. This example adds the `.spec.ttspec.stopManaging` line and adds a value of `Suspend`.

   ```
   kubectl edit timestenscaleout samplescaleout
   # Please edit the object below. Lines beginning with a '#' will be
   ignored,
   # and an empty file will abort the edit. If an error occurs while
   saving this file will be
   # reopened with the relevant failures.
   #
   apiVersion: timesten.oracle.com/v1
   kind: TimesTenScaleout
   metadata:
     creationTimestamp: "2023-01-18T23:57:56Z"
     generation: 1
   ...
   spec
   ...
     ttspec:
       ...
       dbConfigMap:
       - samplescaleout
       ...
       k: 2
       ...
       nReplicaSets: 3
       nZookeeper: 3
       replicaSetRecovery: Restart
       stopManaging: Suspend
   ...
   timestenscaleout.timesten.oracle.com/samplescaleout edited
   ```

2. Use the `kubectl get` command to observe the behavior..

   ```
   kubectl get tts samplescaleout
   NAME              OVERALL                        MGMT     CREATE
   LOAD               OPEN   AGE
   samplescaleout    ManualInterventionRequired    Normal   created
   loaded-complete    open   3h33m
   ```

The Operator sets the TimesTenScaleout object to the `ManualInterventionRequired` High Level state. The Operator takes no further action on the object. You can now perform manual operations on your grid and database. When you have completed such operations and are ready for the Operator to resume management, you can set the `.spec.ttspec.rexamine` datum for the object. For an example that shows you how to set a TimesTenScaleout object's `.spec.ttspec.rexamine` datum, see Set reexamine.

Set reexamine

If a TimesTenScaleout object is in the `ManualInterventionRequired` state, you can set/modify the object's `.spec.ttspec.reexamine` datum to instruct the TimesTen Operator to move the object into the `Reexamine` state. In the `Reexamine` state, the Operator examines the state of TimesTen and the database. If both are healthy, the Operator returns the object to the High Level `Normal` state. If not healthy, the object re-enters the `ManualInterventionRequired` state.

Let's walk through an example that shows you how to reexamine a TimesTenScaleout object.

1. Verify a TimesTenScaleout object is in the `ManualInterventionRequired` state.

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL                     MGMT     CREATE
   LOAD              OPEN   AGE
   samplescaleout   ManualInterventionRequired   Normal   created   loaded-
   complete    open    3h48m
   ```

2. Use the `kubectl edit` command to edit the TimesTenScaleout object, making the following changes:

   - If there is a line for `.spec.ttspec.reexamine` in the file, then modify its value. It must be different than the current value.
   - If there is no line for `.spec.ttspec.reexamine` in the file, then add a line and specify a value.

   In this example, there is no `.spec.ttspec.reexamine` line. This example adds the `.spec.ttspec.reexamine` line and assigns a value of `Reexamine1`.

   ```
   kubectl edit timestenscaleout samplescaleout
   # Please edit the object below. Lines beginning with a '#' will be
   ignored,
   # and an empty file will abort the edit. If an error occurs while saving
   this file will be
   # reopened with the relevant failures.
   #
   apiVersion: timesten.oracle.com/v1
   kind: TimesTenScaleout
   metadata:
     creationTimestamp: "2023-01-18T23:57:56Z"
     generation: 1
   ...
   spec
   ...
     ttspec:
       ...
       dbConfigMap:
       - samplescaleout
       ...
       k: 2
       ...
       nReplicaSets: 3
   ```

```
    nZookeeper: 3
    replicaSetRecovery: Restart
    reexamine: Reexamine1
...
timestenscaleout.timesten.oracle.com/samplescaleout edited
```

3. Use the `kubectl get` command to observe the behavior..

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL     MGMT     CREATE    LOAD
   OPEN   AGE
   samplescaleout   Reexamine   Normal   created   loaded-complete
   open   3h52m
   ```

   The object is in the `Reexamine` High Level state. The Operator examines the state of TimesTen and the database. If healthy, the Operator moves the object to the High Level `Normal` state. If not healthy, the object re-enters the `ManualInterventionRequired` state.

   ```
   kubectl get tts samplescaleout
   NAME             OVERALL   MGMT     CREATE    LOAD
   OPEN   AGE
   samplescaleout   Normal    Normal   created   loaded-complete
   open   3h53m
   ```

The object transitioned to the `Normal` High Level state, indicating the grid and database are functioning normally.

10

Configuring with the TimesTen Prometheus Exporter

This chapter discusses how the TimesTen Operator can configure, start, and manage the TimesTen Exporter. The Exporter can then collect metrics from the TimesTen databases that are running in your Kubernetes cluster, and expose these metrics to Prometheus.

Topics:

About Prometheus and the TimesTen exporter

Prometheus is an open source monitoring and alerting toolkit. It collects and stores metrics from monitored targets by scraping HTTP (or HTTPS) `metrics` endpoints on these targets.

The TimesTen exporter (Exporter) collects TimesTen metrics and exposes these metrics to Prometheus. The Exporter consists of a single executable program (utility) called `ttExporter`. The Exporter presents itself as an HTTP or HTTPS server. When the Exporter receives an HTTP or HTTPS request to `/metrics`, it retrieves the TimesTen metrics from each database that it monitors and prepares a plain text HTTP or HTTPS response with the metrics.

The Exporter is supported in TimesTen Classic and in TimesTen Scaleout. In TimesTen Scaleout, the Exporter is deployed on each data instance and on the management instance.

The `ttExporter` utility is included in the `/bin` directory of a TimesTen instance, and sourcing the `/bin/ttenv` script of that instance will put it on your `PATH`.

The Exporter can be configured either to require client certificate authentication with Transport Layer Security (mutual TLS) or to require no authentication. In a client certificate authentication configuration, an Oracle Wallet is used to store the TLS credentials.

The TimesTen Kubernetes Operator supports the Exporter. It provides facilities to automatically configure, start, and manage this utility in a Kubernetes environment.

For information about the Exporter and the ttExporter utility, see About the TimesTen exporter in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide* and ttExporter in the *Oracle TimesTen In-Memory Database Reference*.

This documentation discusses how to configure the TimesTen Operator with the TimesTen exporter. It does not discuss how to set up and configure Prometheus. For information about Prometheus, see https://prometheus.io/docs/introduction/overview/

About the TimesTen Operator and the TimesTen exporter

The TimesTen Kubernetes Operator supports the TimesTen Exporter and lets you configure your TimesTenClassic or your TimesTenScaleout object to use this Exporter. Once configured, the Exporter collects metrics from the TimesTen databases running in your Kubernetes environment and exposes these metrics to Prometheus.

The `ttExporter` utility runs in its own `exporter` container. This container exists in the same TimesTen Pods as the `tt` and the `daemonlog` containers. These TimesTen Pods run in your Kubernetes cluster.

The TimesTen Operator provisions the containers that are running the Exporter. You configure the Exporter by using the `prometheus` object located in the `.spec.ttspec` definition of the TimesTenClassic and the TimesTenScaleout object types. When you define your TimesTenClassic or your TimesTenScaleout object with the `.spec.ttspec.prometheus` data item specified, the Operator configures your TimesTen Pods with an `exporter` container. For information on the TimesTen Operator object types, see Overview of the TimesTen Operator object types.

The Operator creates the Pods that are running TimesTen with the Kubernetes `shareProcessNamespace` Pod attribute. This attribute enables the Exporter that is running in the `exporter` container to access TimesTen that is running in the `tt` container, both of which are in the TimesTen Pod. For more information about the `shareProcessNamespace` attribute, see https://kubernetes.io/docs/tasks/configure-pod-container/share-process-namespace/

The Operator configures the `exporter` container with the same TimesTen container image as the `tt` and the `daemonlog` containers. If the Exporter fails or exits, Kubernetes destroys the `exporter` container and creates another one to take its place. Kubernetes monitors and manages the `exporter` container. Because the lifecycle of individual containers in a Pod are independent, the Operator ensures that the `ttExporter` command starts after TimesTen. The Operator waits until the TimesTen agent creates the TimesTen instance and waits until the TimesTen main daemon is running in the `tt` container of the TimesTen Pod. The Operator then starts the Exporter in the `exporter` container of the same TimesTen Pod.

Once configured, the Exporter functions as an HTTP (or HTTPS) server. It listens for incoming GET requests and responds to them by gathering a set of metrics from TimesTen. It then returns these metrics as the response to the GET request.

To facilitate the listening process, the Operator creates a Kubernetes headless Service. This Service exposes the port on which the Exporter is listening to the remainder of the Kubernetes cluster. This lets a Prometheus server running in the cluster to fetch the TimesTen metrics from TimesTen and process them.

Both the Exporter and Prometheus can be configured to require no authentication or to require client certificate authentication with Transport Layer Security (mutual TLS). The TimesTen Operator supports both configurations:

- To configure with no authentication, see:
  - About configuring with no authentication
  - Configure with no authentication
- To configure with client certificate certification, see:

– About configuring with client certificate authentication
– Configure with client certificate authentication

About configuring with no authentication

You can configure your TimesTenClassic or your TimesTenScaleout object to require the TimesTen Exporter to be used with no authentication. The TimesTen Operator provides the `prometheus` object as part of the `.spec.ttspec` definition for both the TimesTenClassic and the TimesTenScaleout object types. This `prometheus` data item signals the TimesTen Operator to configure and deploy the Exporter.

The `spec.ttspec.prometheus` object supports the `insecure` attribute. When you set this attribute to `true`, the Operator configures the Exporter with no authentication. The Operator sends the `ttExporter -insecure` command line option to the `ttExporter` utility. This causes the Exporter to process unencrypted HTTP requests, and results in the Exporter being configured with no authentication.

For a detailed example, see Configure with no authentication.

You can also review the following additional references:

- For information about the `prometheus` object, see TimesTenClassicSpecSpecPrometheus and TimesTenScaleoutSpecSpecPrometheus.
- For information about the command line options for the `ttExporter` utility, see ttExporter in the *Oracle TimesTen In-Memory Database Reference*.

Configure with no authentication

This example shows you how to configure a TimesTenClassic object to use the TimesTen Exporter and to require it to be used with no authentication. You can also use this example to configure a TimesTenScaleout object. The steps are the same.

- Define and deploy with no authentication
- Monitor the deployment
- Verify with no authentication

Define and deploy with no authentication

This example assumes you have already created your metadata files and used one of the Kubernetes mechanisms to place these metadata files into the `/ttconfig` directory of the TimesTen containers. See Populating the /ttconfig directory.

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (This example uses `prometheusnoauth`.) The YAML file contains the definitions for the TimesTenClassic object. In this example, the fields that are specific to configuring the Exporter are as follows:
   - `prometheus`: Specify the `prometheus` object in the `.spec.ttspec` definition. This signals the TimesTen Operator to configure the TimesTen Exporter for the TimesTenClassic object.

- `insecure`: Specify the `insecure` attribute of the `.spec.ttspec.prometheus` object and set the value to `true`. A value of `true` signals the Operator to configure the Exporter to require no authentication.
- `port`: Specify the port on which the Exporter listens. The default is port `8888`. This example uses port `9999`.

You also need to specify the following fields:

- `name`: Replace `prometheusnoauth` with the name of your TimesTenClassic object.
- `storageClassName`: Replace `oci` with the name of the storage class in your Kubernetes cluster that is used to allocate Persistent Volumes to hold the TimesTen database.
- `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. (This example assumes a production environment and uses 250Gi for storage. For demonstration purposes, you can use 50Gi of storage.) See the `storageSize` and the `logStorageSize` entries in Table 15-3.
- `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location of the image registry and the name of the image. If you are using the Oracle `container-registry.oracle.com/timesten` registry as the image registry and the `timesten:22.1.1.9.0` image as the container image, no replacement is necessary.
- `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes uses to fetch the TimesTen image.
- `dbConfigMap`: This example uses one ConfigMap (called `prometheusnoauth`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files. This ConfigMap is included in the Projected Volume. The volume is mounted as `/ttconfig` in the TimesTen containers. See Using ConfigMaps and Secrets.

```
vi prometheusnoauth

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: prometheusnoauth
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    imagePullSecret: sekret
    prometheus:
      insecure: true
      port: 9999
    dbConfigMap:
    - prometheusnoauth
```

2. Create the TimesTenClassic object from the contents of the YAML file.

   ```
   kubectl create -f prometheusnoauth.yaml
   ```

   The output is the following:

   ```
   timestenclassic.timesten.oracle.com/prometheusnoauth created
   ```

You have successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your active standby pair of TimesTen Classic databases with the Exporter configured in each TimesTen instance begins, but is not yet complete.

Monitor the deployment

These steps show you how to monitor the deployment of your active standby pair of TimesTen databases.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

   ```
   kubectl get ttc prometheusnoauth
   ```

   The output is the following:

   ```
   NAME               STATE          ACTIVE    AGE
   prometheusnoauth   Initializing   None      25s
   ```

2. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

   ```
   kubectl get ttc prometheusnoauth
   ```

   The output is the following:

   ```
   NAME               STATE    ACTIVE               AGE
   prometheusnoauth   Normal   prometheusnoauth-0   2m38s
   ```

Your active standby pair of TimesTen Classic databases with the Exporter configured in each TimesTen instance is deployed and running in your Kubernetes cluster. The Exporter is configured to require no authentication.

Verify with no authentication

You can verify that the TimesTen exporter is running and is set up correctly. These steps are optional.

1. Verify the TimesTen Operator has created the Kubernetes headless Service for the Exporter. This Service exposes the port on which the Exporter listens, to the Pods in the

Kubernetes cluster. In this example, the port is `9999`. Recall that you specified this port in your TimesTenClassic object definition. See Define and deploy with no authentication.

```
kubectl get Service prometheusnoauth
```

The output is the following:

```
NAME               TYPE        CLUSTER-IP   EXTERNAL-IP
PORT(S)             AGE
prometheusnoauth   ClusterIP   None         <none>        6625/
TCP,9999/TCP   4m14s
```

2. Establish a shell in the container of the Prometheus Pod. For verification and demonstration purposes only, you can also establish a shell in a TimesTen client Pod. This example uses a TimesTen client Pod.

```
kubectl exec -it client -c client -- /bin/bash
```

3. Use the cURL command to expose the TimesTen metrics.

```
curl http://
prometheusnoauth-0.prometheusnoauth.mynamespace.svc.cluster.local:99
99/metrics
```

In this example, metrics are fetched from `prometheusnoauth-0.prometheusnoauth.mynamespace.svc.cluster.local` and from `prometheusnoauth-1.prometheusnoauth.mynamespace.svc.cluster.local`. These are DNS names, where:

- `prometheusnoauth-0` (or `prometheusnoauth-1`) is the name of the TimesTen Pod.
- `prometheusnoauth` is the name of the TimesTenClassic object.
- `mynamespace` is the name of your namespace.
- `svc.cluster.local` completes the required format for the DNS name.

The Exporter has also been configured to listen on port `9999` and to expose the TimesTen metrics at the HTTP `/metrics` endpoint.

This example shows the output of two metrics. In a real life scenario, there are several TimesTen metrics that are collected by the Exporter and exposed by Prometheus.

```
# HELP timesten_rate_limited State of rate limiting of the exporter
# TYPE timesten_rate_limited gauge
timesten_rate_limited{instancename="instance1"} 0
# HELP timesten_ttmesg_file_bytes Number of bytes in ttmesg.log
# TYPE timesten_ttmesg_file_bytes counter
timesten_ttmesg_file_bytes{instancename="instance1"} 304330
...
```

You have successfully verified that the Exporter is running and is set up to require no authentication. The Exporter is collecting the TimesTen metrics in the `exporter` containers of the TimesTen Pods. Prometheus can now expose these metrics. The TimesTen Operator, together with Kubernetes, manages and monitors the entire process.

About configuring with client certificate authentication

You can configure your TimesTenClassic or your TimesTenScaleout object to require the TimesTen Exporter to be used with client certificate authentication.

Here is a summary of the steps you need to complete. There is a complete example in Configure with client certificate authentication.

- Create a TimesTen instance. See Before you begin.
- Use the `ttExporter` utility to generate the certificates. One of the certificates that is created is the self-signed server certificate. This certificate is placed in an Oracle Wallet. See Create the certificates.
- Place the Oracle Wallet into a Kubernetes Secret. See Create the Kubernetes Secret for the Oracle Wallet.
- Specify the name of the Secret in the `certSecret` attribute of the `spec.ttspec.prometheus` object definition. See Define and deploy with client certificate authentication.
- Save the PEM formatted file containing the server certificate, the client certificate, and the client private key that were created when you ran the `ttExporter` utility. You need these later to configure the Prometheus server. See Create the certificates.

Here are additional references:

- For information about the `prometheus` object, see TimesTenClassicSpecSpecPrometheus and TimesTenScaleoutSpecSpecPrometheus.
- For information about the command line options for the `ttExporter` utility, see ttExporter in the *Oracle TimesTen In-Memory Database Reference*.

Configure with client certificate authentication

This example shows you how to configure a TimesTenClassic object to use the TimesTen Exporter and to require it to be used with client certificate authentication. You can also use this example to configure a TimesTenScaleout object. The steps are the same.

- Before you begin
- Create the certificates
- Create the Kubernetes Secret for the Oracle Wallet
- Define and deploy with client certificate authentication
- Monitor the deployment

Before you begin

The `ttExporter` utility is located in the `/bin` directory of a TimesTen instance. Since the `ttExporter` utility is located in the TimesTen instance, you are required to create a TimesTen instance on your development host so that you have access to the `ttExporter` utility. You create a TimesTen instance from a TimesTen installation. A TimesTen installation is created when you unzip the TimesTen distribution.

You must download the TimesTen distribution and unzip it to create a TimesTen installation before beginning these steps. You may have already completed this process if you chose to build the TimesTen container image. See Unpack the TimesTen and the TimesTen Kubernetes Operator distributions.

1. If you have not already done so, download and unzip the TimesTen distribution into a directory on your development host.
2. On your development host from a directory of your choice, create a directory for the TimesTen instance. This example assumes you have previously created the `/scratch/ttuser` directory. The example creates the `/scratch/ttuser/instance1_exporter_dir` directory.

   ```
   mkdir /scratch/ttuser/instance1_exporter_dir
   ```

3. Create the TimesTen instance located in the TimesTen installation directory. Replace the following:
   - *`installation_dir`*: Name of the TimesTen installation directory. This is the directory where you unzipped the TimesTen distribution.
   - `tt22.1.1.9.0`: TimesTen release number in `tt`*`dottedrelease`* format, where *`dottedrelease`* is `22.1.1.9.0` in this example.
   - `instance1_exporter`: Name of the TimesTen instance.
   - `/scratch/ttuser/instance1_exporter_dir`: Location of the TimesTen instance. You created this directory in the previous step.

   ```
   ./installation_dir/tt22.1.1.9.0/bin/ttInstanceCreate -name
   instance1_exporter -location /scratch/ttuser/instance1_exporter_dir
   ```

   The output is similar to the following:

   ```
   Creating instance in /scratch/ttuser/instance1_exporter_dir/
   instance1_exporter ...

   NOTE: The TimesTen daemon startup/shutdown scripts have not been
   installed.

   The startup script is located here :
           '/scratch/ttuser/instance1_exporter_dir/instance1_exporter/
   startup/tt_instance1_exporter'

   Run the 'setuproot' script :
           /scratch/ttuser/instance1_exporter_dir/
   instance1_exporter/bin/setuproot -install
   This will move the TimesTen startup script into its appropriate
   location.

   The 22.1 Release Notes are located here :
     '/scratch/ttuser/installation_dir/tt22.1.1.9.0/README.html'

   Instance created successfully.
   ```

4. Set the `TIMESTEN_HOME` environment variable. You must set this variable before you run the `ttExporter` utility. This example uses the `bash` Bourne-type shell.

   ```
   . /scratch/ttuser/instance1_exporter_dir/instance1_exporter/bin/ttenv.sh
   ```

   The output is similar to the following, with not all output shown:

   ```
   LD_LIBRARY_PATH set to ...
   ...
   PATH set to ...
   ...
   CLASSPATH set to ...
   TIMESTEN_HOME set to /scratch/ttuser/instance1_exporter_dir/
   instance1_exporter
   ```

You successfully created the TimesTen instance on your development host. You are now ready to use the `ttExporter` utility to create the certificates.

Create the certificates

In order for the TimesTen exporter to be configured to require client certificate authentication, you must create the required certificates. You use the `ttExporter` utility to do this. For details on the `ttExporter` utility, see ttExporter in the *Oracle TimesTen In-Memory Database Reference*.

The required certificates are as follows:

- Server certificate: A self-signed certificate that is stored in an Oracle Wallet. This certificate is used by the Exporter. The name of the Oracle Wallet is `cwallet.sso`.
- Exported server certificate: The self-signed server certificate in PEM format. This certificate is required for your Prometheus configuration.
- Client certificate and client private key: The client certificate and the client private key required for your Prometheus configuration.

The following steps show you how to create these certificates:

1. Check that the `TIMESTEN_HOME` environment variable is set. You set this environment variable in a previous step. See Before you begin.

   ```
   echo $TIMESTEN_HOME
   ```

   The output is the following:

   ```
   /scratch/ttuser/instance1_exporter_dir/instance1_exporter
   ```

2. On your development host, from a directory of your choice, create a subdirectory to store the Oracle Wallet. This example creates the `exportercertdir` directory.

   ```
   mkdir -p exportercertdir
   ```

3. Create the self-signed server certificate. This certificate is stored as an Oracle Wallet. The name of the file that contains the Oracle Wallet is `cwallet.sso`. It contains the certificate information required by the TimesTen Exporter. Later, you will use a

Kubernetes Secret to place the `cwallet.sso` Oracle Wallet file into the `/ttconfig/exporterWallet` location of the `exporter` container.

```
ttExporter -create-server-certificate -certificate-common-name
*.prometheusauth.mynamespace.svc.cluster.local -certificate-alt-
names *.prometheusauth.mynamespace.svc.cluster.local -certificate-
directory exportercertdir
```

The `-certificate-common-name` and `-certificate-alt-names` `ttExporter` options are required. For detailed information on these options, see ttExporter in the *Oracle TimesTen In-Memory Database Reference*.

The `-certificate-common-name` option is the Common Name (CN) that is included in the certificate. It matches the DNS name where the certificate is installed. This CN can contain only one name. Single-level wildcards are acceptable. In this example, the CN name is `*.prometheusauth.mynamespace.svc.cluster.local`, where:

- `*` is a single level wildcard.
- `prometheusauth` is the name of your TimesTenClassic or your TimesTenScaleout object.
- *`mynamespace`* is the name of your namespace.
- `svc.cluster.local` completes the required format for the DNS name.

The `-certificate-alt-names` option is the Subject Alternative Name (SAN) that is included in the certificate. This name includes the CN mentioned previously as well as any other DNS names that need access to the TimesTen Exporter. Single level wildcards are acceptable. In this example, the SAN name includes only the CN name. Specifically, the SAN name is `*.prometheusauth.mynamespace.svc.cluster.local`, where:

- `*` is a single level wildcard.
- `prometheusauth` is the name of your TimesTenClassic or your TimesTenScaleout object.
- *`mynamespace`* is the name of your namespace.
- `svc.cluster.local` completes the required format for the DNS name.

Since these options require you to specify the name of the TimesTenClassic (or the TimesTenScaleout) object and the name of your namespace, you must know these names before completing this step. In addition, you must use these same names when defining your TimesTen Classic or your TimesTenScaleout object.

4. Export the server certificate.

```
ttExporter -export-server-certificate exportercertdir/server.crt -
certificate-directory exportercertdir
```

This command exports the server certificate in PEM format. In this example, the name of the file that contains the certificate is `server.crt`. Save this file. You need it later when configuring Prometheus.

5. Create and export the client client certificate and the client private key.

   ```
   ttExporter -export-client-certificate exportercertdir/client.crt -export-
   client-private-key exportercertdir/key.crt -certificate-directory
   exportercertdir
   ```

   This command creates the client certificate. In this example, the contents of the client certificate is stored in the `client.crt` file. The example also creates the client private key and stores its contents in the `key.crt` file. Save these files. You need them later when configuring Prometheus.

6. Optional: Verify the `ttExporter` utility has created the certificates.

   ```
   ls -a exportercertdir
   ```

   The output is similar to the following:

   ```
   .   client.crt  server.crt
   ..  key.crt     .ttwallet.BA0F2D86-B6D2-4095-A4D0-CDF1FF89E9BF
   ```

   Verify the `ttExporter` utility has created the Oracle Wallet.

   ```
   ls -a exportercertdir/.ttwallet*
   ```

   The output is the following:

   ```
   .  ..  cwallet.sso
   ```

You have successfully created the server certificate, the client certificate, and the client private key. Make a note of these files and their location. You need them later. Specifically you need to specify the `cwallet.sso` Oracle Wallet file when you create the Kubernetes Secret. See Create the Kubernetes Secret for the Oracle Wallet. In addition, you need to specify the `server.crt`, the `client.crt`, and the `key.crt` files later when you configure Prometheus.

> **Note:**
>
> Configuring Prometheus is outside the scope of this book. For information on configuring Prometheus, see About configuring the TimesTen exporter and Prometheus with client certificate authentication in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

Create the Kubernetes Secret for the Oracle Wallet

The following steps show you how to create the Kubernetes Secret for the Oracle Wallet. This Oracle Wallet contains the self-signed server certificate. You created the Oracle Wallet in Create the certificates.

1. On your development host, from a directory of your choice, create an empty subdirectory for the Oracle Wallet (the `cwallet.sso` file). This example creates the `walletdir` subdirectory.

   ```
   mkdir -p walletdir
   ```

2. Copy the `cwallet.sso` Oracle Wallet to the directory.

   ```
   cp exportercertdir/.ttwallet*/cwallet.sso walletdir/cwallet.sso
   ```

   In this example, the Oracle Wallet is located in the `exportercertdir/.ttwallet*/cwallet.sso walletdir` directory. You created this directory in Create the certificates.

3. Create the Kubernetes Secret for the Oracle Wallet. Ensure to specify the `/exporterWallet` directory.

   ```
   kubectl create secret generic prometheuscert --from-
   file=exporterWallet=walletdir/cwallet.sso
   ```

   The `kubectl create generic secret` command does the following:

   - Creates the `prometheuscert` Kubernetes Secret.
   - Includes the `exporterWallet` metadata file. This file is required when including the `cwallet.sso` file in the Secret.
   - Defines `walletdir` as the location for the `cwallet.sso` file.
   - Defines the `cwallet.sso` file as the name of the Oracle Wallet file.

   The output is the following:

   ```
   secret/prometheuscert created
   ```

You have successfully created the Kubernetes Secret. Make a note of the name of the Secret. You use it later when you create your TimesTenClassic or TimesTenScaleout object.

Define and deploy with client certificate authentication

This example shows you how to define a TimesTenClassic object that configures the TimesTen Exporter to require client certificate authentication. Use the same approach for a TimesTenScaleout object. The steps are the same.

This example assumes you have already created your Kubernetes Secret for the Oracle Wallet that contains the self-signed server certificate. See Create the Kubernetes Secret for the Oracle Wallet. In addition, the example assumes that you have created any other metadata files and used one of the Kubernetes mechanisms to place these metadata files into the `/ttconfig` directory of the TimesTen containers. See Populating the /ttconfig directory.

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (This example uses `prometheusauth`.) The YAML file contains the definitions for the

TimesTenClassic object. In this example, the fields that are specific to configuring the Exporter are as following:

- `prometheus`: Specify the `prometheus` object in the `.spec.ttspec` definition. This signals the TimesTen Operator to configure the TimesTen Exporter for the TimesTenClassic object.
- `certSecret`: Specify the `certSecret` attribute of the `.spec.ttspec.prometheus` object. Supply the name of the Kubernetes Secret that contains the Oracle Wallet. You created this Kubernetes Secret in Create the Kubernetes Secret for the Oracle Wallet, and you named the Secret `prometheuscert`. For example, `certSecret: prometheuscert`
- `port`: Specify the port on which the Exporter listens. The default is port `8888`. This example uses port `7777`.

You also need to specify the following fields:

- `name`: Replace `prometheusauth` with the name of your TimesTenClassic object.
- `storageClassName`: Replace `oci` with the name of the storage class in your Kubernetes cluster that is used to allocate PersistentVolumes to hold the TimesTen database.
- `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. (This example assumes a production environment and uses 250Gi for storage. For demonstration purposes, you can use 50Gi of storage.) See the `storageSize` and the `logStorageSize` entries in Table 15-3.
- `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location of the image registry and the name of the image. If you are using the Oracle `container-registry.oracle.com/timesten` registry as the image registry and the `timesten:22.1.1.9.0` image as the container image, no replacement is necessary.
- `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes uses to fetch the TimesTen image.
- `dbConfigMap`: This example uses one ConfigMap (called `prometheusauth`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files. This ConfigMap is included in the Projected Volume. The volume is mounted as `/ttconfig` in the TimesTen containers. See Using ConfigMaps and Secrets.

```
vi prometheuswithauth

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: prometheusauth
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    prometheus:
      certSecret: prometheuscert
      port: 7777
    dbConfigMap:
    - prometheusauth
```

2. Create the TimesTenClassic object from the contents of the YAML file.

   ```
   kubectl create -f prometheusauth.yaml
   ```

   The output is the following:

   ```
   timestenclassic.timesten.oracle.com/prometheusauth created
   ```

You have successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your active standby pair of TimesTen Classic databases with the Exporter configured in each TimesTen instance begins, but is not yet complete.

Monitor the deployment

These steps show you how to monitor the deployment of your active standby pair of TimesTen databases.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

   ```
   kubectl get ttc prometheusauth
   ```

   The output is the following:

   ```
   NAME             STATE          ACTIVE   AGE
   prometheusauth   Initializing   None     16s
   ```

2. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

   ```
   kubectl get ttc prometheusauth
   ```

   The output is the following:

   ```
   NAME             STATE    ACTIVE             AGE
   prometheusauth   Normal   prometheusauth-0   3m1s
   ```

Your active standby pair of TimesTen Classic databases with the Exporter configured in each TimesTen instance is deployed and running in your Kubernetes cluster. The Exporter that is running in the `exporter` container of each TimesTen Pod functions as an HTTPS server.

11

Working with TimesTen Cache

The TimesTen Operator supports the use of TimesTen Cache with TimesTen Classic and TimesTen Scaleout in your Kubernetes environment.

About using TimesTen Cache

The TimesTen Operator provides interfaces that you can use in TimesTen Classic and TimesTen Scaleout to configure cache groups.

These are the cache-related metadata files that you can provide:

- `cacheUser`: This file contains the TimesTen cache manager user. Its format is `user/ttpwd/orapwd`, containing the username, TimesTen password, and Oracle password for the user. The Oracle user (called the cache administration user in Oracle Database) must exist prior to creating and deploying TimesTen Classic or TimesTen Scaleout. The Operator creates the TimesTen user in the the TimesTen database with the given name and password. The Operator also grants this user the appropriate privileges.

  Here is an example:

  ```
  cachemanageruser/ttmgrpwd/oramgrpwd
  ```

  See cacheUser for more information.

- `cachegroups.sql`: This file may contain `CREATE CACHE GROUP` statements as well as `LOAD CACHE GROUP` statements for the Operator to automatically run when the TimesTen database is created. The file also contains TimesTen built-in procedures to update statistics on the cache group tables (such as, `ttOptEstimateStats` and `ttOptUpdateStats`).

  Here is an example:

  ```
  CREATE READONLY CACHE GROUP readcache
  AUTOREFRESH
    INTERVAL 5 SECONDS
  FROM oratt.readtab (
    keyval NUMBER NOT NULL PRIMARY KEY,
    str VARCHAR2(32)
  );

  LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
  ```

  See cachegroups.sql for more information.

If you provide the `cacheUser` and `cachegroups.sql` files, the Operator uses them to provision TimesTen Cache when a new database is created.

The following metadata files are also relevant for TimesTen Cache:

- `tnsnames.ora`: This file is required. It defines Oracle Net Services that applications connect to. For TimesTen Cache, this file configures the connectivity between the TimesTen and the Oracle Database (from which data is being cached). In this context, TimesTen is the application that is the connection to the Oracle Database. See tnsnames.ora for details.
- `sqlnet.ora`: This file is optional. However, it may be necessary depending on your Oracle Database configuration. The file defines options for how client applications communicate with the Oracle Database. In this context, TimesTen is the application. The `tnsnames.ora` and `sqlnet.ora` files together define how an application communicates with the Oracle Database.

  See sqlnet.ora for details.
- `db.ini`: This file is required. Its contents contain TimesTen connection attributes for your TimesTen database. The files are included in TimesTen's `sys.odbc.ini` file in TimesTen Classic or in the database definition file (dbDef) in TimesTen Scaleout. You must specify the `OracleNetServiceName` and the `DatabaseCharacterSet` connection attributes in this file. The `DatabaseCharacterSet` value must match the value of the Oracle database character set value.

  See db.ini for details.
- `schema.sql`: This file may be required. In TimesTen Cache, one or more cache table users own the cache tables. If this cache table user is not the cache manager user, then you must specify the `schema.sql` file and in it include the schema user. You must also assign the appropriate privileges to this schema user. For example, if the `oratt` schema user was created in the Oracle Database, and this user is not the TimesTen cache manager user, you must create the TimesTen `oratt` user in this file.

  The instance administrator uses the `ttIsql` utility to run this file immediately after the database is created. This file is run before the Operator configures TimesTen Cache, so ensure there are no cache definitions in this file.

  See Create the Oracle Database users for more information on the schema users in the Oracle Database. See schema.sql for details about the `schema.sql` file.

In TimesTen Classic, the contents of the `cachegroups.sql` file runs on the active database before it is duplicated to the standby. If there are autorefresh cache groups specified in the `cachegroups.sql` file, they are paused by the agent prior to duplicating the active database to the standby. After the duplication process completes, these autorefresh cache groups are re-enabled.

In TimesTen Scaleout, the contents of the `cachegroups.sql` file runs during database creation.

Once created and rolled out, the Operator does not monitor or manage TimesTen Cache. Specifically, the Operator does not monitor the health of the cache agents, nor does it take further action to start or stop them. In addition, the Operator does not verify that data is propagating correctly between the TimesTen database and the Oracle Database.

If you delete your TimesTenClassic or TimesTenScaleout object, the Operator automatically cleans up the Oracle Database metadata. If, however, you want to retain the Oracle Database metadata, specify the `cacheCleanUp` datum in your TimesTenClassic or TimesTen Scaleout object definition and set its value to `false`.

See `cacheCleanup` datum in TimesTenClassicSpecSpec and TimesTenScaleoutSpecSpec.

For complete examples, see the following:

- For TimesTen Classic deployments: See TimesTen Cache in TimesTen Classic Example.
- For TimesTen Scaleout deployments: See TimesTen Cache in TimesTen Scaleout Example.

12

Using Encryption for Data Transmission

TimesTen replication and TimesTen Client/Server support the use of Transport Layer Security (TLS) for communication between TimesTen instances.

This chapter details the process for configuring and using TLS in your Kubernetes environment. This enables encrypted data transmission between your replicated TimesTen databases and, if in a Client/Server environment, between your TimesTen client applications and your TimesTen Server (your TimesTen database).

Topics include:

- Creating TLS certificates for replication and Client/Server
- Configuring TLS for replication
- Configuring TLS for Client/Server

Creating TLS certificates for replication and Client/Server

By default, TimesTen replication transmits data between your TimesTen databases unencrypted. In addition, in a TimesTen Client/Server environment, by default data is transmitted unencrypted between your application and your TimesTen database.

You can choose to enable encryption for replication and for Client/Server through the use of Transport Layer Security (TLS). TimesTen provides the `ttCreateCerts` utility to generate self-signed certificates for TLS. For more information on TLS certificates and wallets, see About using certificates with TimesTen in the *Oracle TimesTen In-Memory Database Security Guide*.

Note:

Java must be installed on your development host in order for you to use the `ttCertsCreate` utility. The utility searches for Java according to the `JRE_HOME`, `JAVA_HOME`, and `PATH` settings.

The `ttCreateCerts` utility is located in the `/bin` directory of a TimesTen instance. The utility creates three wallets: `rootWallet`, `clientWallet`, and `serverWallet`.

From your Linux development host, perform these steps to create the certificates.

1. Navigate to the `bin` directory of the installation and run the `ttInstanceCreate` utility interactively to create an instance. Recall that the *`installation_dir`* directory was created when you unpacked the TimesTen distribution. See "Unpack the TimesTen and the TimesTen Kubernetes Operator distributions" for information on unpacking the TimesTen distribution.

   You have to create a TimesTen instance as the `ttCreateCerts` utility is run from a TimesTen instance. For more information on the `ttInstanceCreate` utility, see "ttInstanceCreate" in the *Oracle TimesTen In-Memory Database Reference*.

Create the instance directory (`/scratch/ttuser/instance_dir`, in this example), then run the `ttInstanceCreate` utility, supplying the `-name` and the `-location` parameters. This example uses `instance1` as the name of the instance and uses `/scratch/ttuser/instance_dir` as the location of the instance.

```
% mkdir /scratch/ttuser/instance_dir

% installation_dir/tt22.1.1.9.0/bin/ttInstanceCreate -name instance1
-location /scratch/ttuser/instance_dir
Creating instance in /scratch/ttuser/instance_dir/instance1 ...
INFO: Mapping files from the installation to /scratch/ttuser/
instance_dir/instance1/install

NOTE: The TimesTen daemon startup/shutdown scripts have not been installed.

The startup script is located here :
        '/scratch/ttuser/instance_dir/instance1/startup/tt_instance1'

Run the 'setuproot' script :
        /scratch/ttuser/instance_dir/instance1/bin/setuproot -install
This will move the TimesTen startup script into its appropriate location.

The 22.1 Release Notes are located here :
  'installation_dir/tt22.1.1.9.0/README.html'
```

2. Set the `TIMESTEN_HOME` environment variable. This variable must be set before you run the `ttCertsCreate` utility. From the `bin` directory of the instance, source the `ttenv.csh` or the `ttenv.sh` script.

   This example uses the `bash` Bourne-type shell. (Not all output is shown.)

```
% . /scratch/ttuser/instance_dir/instance1/bin/ttenv.sh
LD_LIBRARY_PATH set to
...
PATH set to
...
CLASSPATH set to
TIMESTEN_HOME set to /scratch/ttuser/instance_dir/instance1
```

3. Run the `ttCreateCerts` utility from the `bin` directory of the instance. This example uses the `-verbose` qualifier to show detailed output. See "Generation of certificates for TimesTen Classic" in the *Oracle TimesTen In-Memory Database Security Guide* for more information on the `ttCreateCerts` utility.

   The default wallet directory is *`timesten_home`*`/conf`, where *`timesten_home`* is the TimesTen instance home directory. This example uses this default wallet directory.

```
% /scratch/ttuser/instance_dir/instance1/bin/ttCreateCerts -verbose
Requested Certificates:
User Certificates:
Subject:        CN=server1,C=US
Trusted Certificates:
Subject:        CN=ecRoot,C=US
Requested Certificates:
User Certificates:
Subject:        CN=client1,C=US
Trusted Certificates:
Subject:        CN=ecRoot,C=US
ttCreateCerts : certificates created in /scratch/ttuser/instance_dir/
instance1/conf
```

4. Review the wallet locations and the certificates (represented in **bold**). The `cwallet.sso` in the `serverWallet` directory is the file you will supply as the `replicationWallet` metadata file for replication and for the server in a Client/Server environment. The `cwallet.sso` in the `clientWallet` directory is the file you will use for the client in a Client/Server environment. See "Configuration metadata details" for information on the `replicationWallet` and the `clientWallet` metadata files. Also see "Configuring TLS for replication" and "Configuring TLS for Client/Server" for information on using these metadata files.

   (These `cwallet.sso` files are also represented in **bold**).

```
% ls $TIMESTEN_HOME/conf
client1.cert  root.cert    server1.cert  snmp.ini      sys.ttconnect.ini
clientWallet  rootWallet   serverWallet  sys.odbc.ini  timesten.conf

% ls $TIMESTEN_HOME/conf/*Wallet*
/scratch/ttuser/instance_dir/instance1/conf/clientWallet:
cwallet.sso  cwallet.sso.lck

/scratch/ttuser/instance_dir/instance1/conf/rootWallet:
cwallet.sso  cwallet.sso.lck

/scratch/ttuser/instance_dir/instance1/conf/serverWallet:
cwallet.sso  cwallet.sso.lck
```

You have successfully created the certificates that can be used for TLS for both replication and TimesTen Client/Server. You are now ready to configure and use TLS for replication, for Client/Server, or for both replication and Client/Server.

Configuring TLS for replication

You can configure TLS for replication to ensure secure network communication between your replicated TimesTen databases. See "Transport Layer Security for TimesTen replication" in the *Oracle TimesTen In-Memory Database Security Guide* for detailed information.

These sections describe how to configure and use TLS for replication:

- Create the metadata files and the Kubernetes facilities
- Create the TimesTenClassic object
- Monitor the deployment of the TimesTenClassic object
- Verify that TLS is being used for replication

Create the metadata files and the Kubernetes facilities

The `/ttconfig/replicationWallet` metadata file is required for TLS support for replication. (The `/ttconfig` directory is located in the containers of your TimesTen databases.) This file must contain the `cwallet.sso` file (the Oracle wallet) that was generated when you created the TLS certificates. Recall that this file was located in the `/scratch/ttuser/instance_dir/instance1/conf/serverWallet` directory. See Creating TLS certificates for replication and Client/Server for information on creating these certificates. This wallet contains the credentials that are used by TimesTen replication for configuring TLS encryption between your active standby pair of TimesTen databases.

In addition to the `/ttconfig/replicationWallet` metadata file, you may use the other supported metadata files. See Configuration metadata details for information on these supported metadata files.

You can include these metadata files in one or more Kubernetes facilities (for example, in a Kubernetes Secret, in a ConfigMap, or in an init container). This ensures the metadata files are populated in the `/ttconfig` directory of the TimesTen containers. Note that there is no requirement as to how to get the metadata files into this `/ttconfig` directory. See Populating the /ttconfig directory for more information.

The example in the following sections illustrates how to include the `replicationWallet` metadata file in a Kubernetes Secret. It also creates the `db.ini`, the `adminUser`, and the `schema.sql` metadata files and includes these metadata files in a ConfigMap:

- Create the Kubernetes Secret
- Create the ConfigMap

Create the Kubernetes Secret

This section creates the `repl-tls` Kubernetes Secret. The `repl-tls` Secret will contain the `replicationWallet` metadata file.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory. This example creates the `serverWallet` subdirectory. (The `serverWallet` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p serverWallet
   ```

2. Copy the `/scratch/ttuser/instance_dir/instance1/conf/serverWallet/cwallet.sso` file into the `serverWallet` directory that you just created. Recall that this file was generated when you used the `ttCreateCerts` utility to create the TLS certificates. See "Creating TLS certificates for replication and Client/Server" for information.

   ```
   % cp /scratch/ttuser/instance_dir/instance1/conf/serverWallet/cwallet.sso
   serverWallet/cwallet.sso
   ```

3. Create the Kubernetes Secret.

   In this example:

   - The name of the Secret is `repl-tls`. Replace `repl-tls` with a name of your choosing. (`repl-tls` is represented in **bold**.)
   - The name of the metadata file required for TLS replication is `replicationWallet` (represented in **bold**).
   - The location of the wallet directory is `serverWallet` (in this example, represented in **bold**). If you use a different directory, replace `serverWallet` with the name of your directory.
   - The name of the Oracle wallet is `cwallet.sso` (represented in **bold**).

   Use the `kubectl create` command to create the Secret:

   ```
   % kubectl create secret generic repl-tls
   --from-file=replicationWallet=serverWallet/cwallet.sso
   secret/repl-tls created
   ```

You have successfully created and deployed the `repl-tls` Kubernetes Secret. The `replicationWallet/cwallet.sso` file will later be available in the `/ttconfig` directory of the TimesTen containers. In addition, the file will be available in the `/tt/home/timesten/replicationWallet` directory of the TimesTen containers.

Create the ConfigMap

This section creates the `repl-tls` ConfigMap. This ConfigMap contains the `db.ini`, the `adminUser`, and the `schema.sql` metadata files.

These metadata files are not required for TLS, but are included as additional attributes for your TimesTen databases. See "Overview of the configuration metadata and the Kubernetes facilities" for information on the metadata files and the ConfigMap facility.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_replTLS` subdirectory. (The `cm_replTLS` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p cm_replTLS
   ```

2. Navigate to the ConfigMap directory.

   ```
   % cd cm_replTLS
   ```

3. Create the `db.ini` file in this ConfigMap directory (`cm_replTLS`, in this example). In this `db.ini` file, define the `PermSize` and `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   ```

4. Create the `adminUser` file in this ConfigMap directory (`cm_replTLS`, in this example). In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

5. Create the `schema.sql` file in this ConfigMap directory (`cm_replTLS`, in this example). In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator will automatically initialize your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

6. Create the ConfigMap. The files in the `cm_replTLS` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

   In this example:

- The name of the ConfigMap is `repl-tls`. Replace `repl-tls` with a name of your choosing. (`repl-tls` is represented in **bold** in this example.)
- This example uses `cm_replTLS` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_replTLS` with the name of your directory. (`cm_replTLS` is represented in **bold** in this example.)

Use the `kubectl create` command to create the ConfigMap:

```
% kubectl create configmap repl-tls --from-file=cm_replTLS
configmap/repl-tls created
```

7. Use the `kubectl describe` command to verify the contents of the ConfigMap. (`repl-tls`, in this example.)

```
% kubectl describe configmap repl-tls
Name:         repl-tls
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
adminUser:
----
sampleuser/samplepw

db.ini:
----
PermSize=200
DatabaseCharacterSet=AL32UTF8

schema.sql:
----
create sequence sampleuser.s;
create table sampleuser.emp (id number not null primary key, name char (32));

Events:  <none>
```

You have successfully created and deployed the `repl-tls` ConfigMap.

Create the TimesTenClassic object

This section creates the TimesTenClassic object. See "Defining and creating the TimesTenClassic object" and "About the TimesTenClassic object type" for detailed information on the TimesTenClassic object.

Perform these steps:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (In this example, `repltls`.) The YAML file contains the definitions for the TimesTenClassic object. See "TimesTenClassicSpecSpec" for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   In this example, the fields of particular interest for TLS replication are:

   - `dbSecret`: This example uses one Kubernetes Secret (called `repl-tls`) for the `replicationWallet` metadata file.

- `replicationCipherSuite`: This field is required for TLS for replication. In this example, the value is `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`. See "Configuration for TLS for replication" in the *Oracle TimesTen In-Memory Database Security Guide* and see the `replicationCipherSuite` entry in "Table 15-3" in this book for more information.
- `replicationSSLMandatory`: This field is optional. In this example, set `replicationSSLMandatory` equal to 1. See "Configuration for TLS for replication" in the *Oracle TimesTen In-Memory Database Security Guide* and see the `replicationSSLMandatory` entry in "Table 15-3" in this book for more information.

In addition, this example includes:

- `name`: Replace `repltls` with the name of your TimesTenClassic object.
- `storageClassName`: Replace `oci` with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
- `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. Note: This example assumes a production environment and uses a value of `250Gi` for `storageSize`. For demonstration purposes, a value of `50Gi` is adequate. See the `storageSize` and the `logStorageSize` entries in "Table 15-3" for information.
- `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location and the name of image.
- `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.
- `dbConfigMap`: This example uses one ConfigMap (called `repl-tls`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files.

```
% vi repltls.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: repltls
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - repl-tls
    dbSecret:
    - repl-tls
    replicationCipherSuite: SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    replicationSSLMandatory: 1
```

2. Use the `kubectl create` command to create the TimesTenClassic object from the contents of the YAML file (in this example, `repltls.yaml`). Doing so begins the process of deploying your active standby pair of TimesTen databases in the Kubernetes cluster.

```
% kubectl create -f repltls.yaml
timestenclassic.timesten.oracle.com/repltls created
```

You have successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your TimesTen databases begins, but is not yet complete.

Monitor the deployment of the TimesTenClassic object

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the active standby pair as it is provisioned.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

   ```
   % kubectl get ttc repltls
   NAME      STATE          ACTIVE   AGE
   repltls   Initializing   None     50s
   ```

2. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

   ```
   % kubectl get ttc repltls
   NAME      STATE     ACTIVE      AGE
   repltls   Normal    repltls-0   3m45s
   ```

3. Use the `kubectl describe` command to view the active standby pair provisioning in detail.

   Note the following have been correctly set in the `repltls` TimesTenClassic object definition:

   - The `repl-tls` Secret has been correctly referenced in the `dbSecret` field (represented in **bold**).
   - The `repl-tls` Configmap has been correctly referenced in the `dbConfigMap` field (represented in **bold**).
   - The `replicationCipherSuite` field has been correctly set to `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` (represented in **bold**).
   - The `replicationSSLMandatory` field has been correctly set to `1` (represented in **bold**).

   Note: Not all of the output is shown in this example.

   ```
   % kubectl describe ttc repltls
   Name:         repltls
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   API Version:  timesten.oracle.com/v1
   Kind:         TimesTenClassic
   Metadata:
     Creation Timestamp:  2022-04-30T18:51:43Z
     Generation:          1
     Resource Version:    75029797
     Self Link:
   /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/repltls
     UID:                 a2915ef3-0fe0-11eb-8b9a-aaa0151611fe
   Spec:
     Ttspec:
       Db Config Map:
         repl-tls
       Db Secret:
         repl-tls
   ```

```
    Image:                        container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Image Pull Policy:            Always
    Image Pull Secret:            sekret
    Replication Cipher Suite:     SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    Replication SSL Mandatory:    1
    Storage Class Name:           oci
    Storage Size:                 250Gi
...
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     Create       4m17s  ttclassic  Secret tta2915ef3-0fe0-11eb-8b9a-
aaa0151611fe created
  -     Create       4m17s  ttclassic  Service repltls created
  -     Create       4m17s  ttclassic  StatefulSet repltls created
  -     StateChange  3m10s  ttclassic  Pod repltls-1 Agent Up
  -     StateChange  3m10s  ttclassic  Pod repltls-1 Release 22.1.1.9.0
  -     StateChange  3m10s  ttclassic  Pod repltls-1 Daemon Up
  -     StateChange  2m3s   ttclassic  Pod repltls-0 Agent Up
  -     StateChange  2m3s   ttclassic  Pod repltls-0 Release 22.1.1.9.0
  -     StateChange  2m1s   ttclassic  Pod repltls-0 Daemon Up
  -     StateChange  68s    ttclassic  Pod repltls-0 Database Loaded
  -     StateChange  68s    ttclassic  Pod repltls-0 Database Updatable
  -     StateChange  68s    ttclassic  Pod repltls-0 CacheAgent Not Running
  -     StateChange  68s    ttclassic  Pod repltls-0 RepAgent Not Running
  -     StateChange  67s    ttclassic  Pod repltls-0 RepState IDLE
  -     StateChange  67s    ttclassic  Pod repltls-0 RepScheme None
  -     StateChange  66s    ttclassic  Pod repltls-0 RepAgent Running
  -     StateChange  66s    ttclassic  Pod repltls-0 RepScheme Exists
  -     StateChange  66s    ttclassic  Pod repltls-0 RepState ACTIVE
  -     StateChange  47s    ttclassic  Pod repltls-1 Database Loaded
  -     StateChange  47s    ttclassic  Pod repltls-1 Database Not Updatable
  -     StateChange  47s    ttclassic  Pod repltls-1 CacheAgent Not Running
  -     StateChange  47s    ttclassic  Pod repltls-1 RepAgent Not Running
  -     StateChange  47s    ttclassic  Pod repltls-1 RepScheme Exists
  -     StateChange  47s    ttclassic  Pod repltls-1 RepState IDLE
  -     StateChange  41s    ttclassic  Pod repltls-1 RepAgent Running
  -     StateChange  36s    ttclassic  Pod repltls-1 RepState STANDBY
  -     StateChange  36s    ttclassic  TimesTenClassic was Initializing, now Normal
```

Your active standby pair of TimesTen databases are successfully deployed (as indicated by `Normal`.) You are now ready to verify that TLS is being used for replication.

Verify that TLS is being used for replication

To verify TLS is being used for replication, perform the following steps:

1. Review the active (`repltls-0`, in this example) pod and the standby pod (`repltls-1`, in this example).

   ```
   % kubectl get pods
   NAME                                  READY   STATUS    RESTARTS   AGE
   repltls-0                             2/2     Running   0          6m35s
   repltls-1                             2/2     Running   0          6m34s
   timesten-operator-f84766548-tch7s     1/1     Running   0          28d
   ```

2. Optional: Use the `kubectl exec -it` command to invoke the shell in the active Pod (`repltls-0`, in this example).

   ```
   % kubectl exec -it repltls-0 -c tt -- /bin/bash
   ```

3. Optional: From the shell in the active pod, verify the `cwallet.sso` file is located in the `/tt/home/timesten/replicationWallet` directory.

   ```
   % ls /tt/home/timesten/replicationWallet
   cwallet.sso
   ```

4. Optional: From the shell in the active pod, verify that the TLS replication-specific values are correct in the `timesten.conf` configuration file. (This file is located in the `/tt/home/timesten/instances/instance1/conf` directory.)

   In particular, note that:

   - `replication_wallet` is correctly set to `/tt/home/timesten/replicationWallet` (represented in **bold**).
   - `replication_cipher_suite` is correctly set to `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` (represented in **bold**).
   - `replication_ssl_mandatory` is correctly set to `1` (represented in **bold**).

   See "Configuration for TLS for replication" in the *Oracle TimesTen In-Memory Database Security Guide* for more information on these `timesten.conf` attributes.

   ```
   % cat /tt/home/timesten/instances/instance1/conf/timesten.conf
   admin_uid=3429
   admin_user=timesten
   daemon_port=6624
   group_name=timesten
   hostname=repltls-0
   instance_guid=48AC5964-56A1-4C66-AB89-5646A2431EA3
   instance_name=instance1
   replication_cipher_suite=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
   replication_ssl_mandatory=1
   replication_wallet=/tt/home/timesten/replicationWallet
   server_port=6625
   show_date=1
   timesten_release=22.1.1
   tns_admin=/ttconfig
   verbose=1
   ```

5. From the shell in the active pod, run the `ttRepAdmin` utility with the `-showstatus -detail` options to verify the replication agent transmitters and receivers are using TLS (as indicated by SSL, represented in **bold**). See "ttRepAdmin" in the *Oracle TimesTen In-Memory Database Reference* for information on this utility.

   Note: Not all output is shown in this example.

   ```
   % ttRepAdmin -showstatus -detail repltls

   Replication Agent Status as of: 2022-04-30 19:01:55

   DSN                          : repltls
   ...
   TRANSMITTER thread(s) (TRANSMITTER(M):139870727366400):
    For                      : REPLTLS (track 0) (SSL)
      Start/Restart count    : 1
      Current state          : STATE_META_PEER_INFO

   RECEIVER thread(s) (RECEIVER:139870719887104):
    For                      : REPLTLS (track 0) (SSL)
      Start/Restart count    : 1
   ```

```
    Current state         : STATE_RCVR_READ_NETWORK_LOOP
...
```

You have successfully verified that TLS for replication is being used.

Configuring TLS for Client/Server

You can configure TLS for Client/Server to ensure secure network communication between TimesTen clients and servers. See "Transport Layer Security for TimesTen Client/Server" in the Oracle TimesTen In-Memory Database Security Guide for detailed information.

There are both server-side and client-side configuration requirements for using TLS for Client/Server. These requirements are detailed in these sections:

- Configuration on the server
- Configuration on the client

Configuration on the server

These sections discuss the configuration requirements for the server. The sections also include an example of how to configure TLS for the server in your Kubernetes cluster.

- Overview of the metadata files and the Kubernetes facilities
- Create the Kubernetes Secret for the csWallet metadata file
- Create the ConfigMap for the server-side attributes
- Create the TimesTenClassic object
- Monitor the deployment of the TimesTenClassic object

Overview of the metadata files and the Kubernetes facilities

The `/ttconfig/csWallet` metadata file is required for TLS support for Client/Server. (The `/ttconfig` directory is located in the containers of your TimesTen databases.) This file must contain the `cwallet.sso` file (the Oracle wallet) that was generated when you created the TLS certificates. This file is the Oracle wallet required for the server. Recall that this file was located in the `/scratch/ttuser/instance_dir/instance1/conf/serverWallet` directory. See Creating TLS certificates for replication and Client/Server for information on creating these certificates. This wallet contains the credentials that are used for configuring TLS encryption between your TimesTen database and your Client/Server applications.

There are also server-side connection attributes that must be set. You can define these attributes in the `db.ini` metadata file. After the `db.ini` file is placed in the `/ttconfig` directory of the TimesTen containers, the Operator copies the contents of the `db.ini` file to the `*timesten_home*/conf/sys.odbc.ini` file located in the TimesTen containers. (Note that `*timesten_home*` is the TimesTen instance directory. This instance directory is `/tt/home/timesten/instances/instance1` in your TimesTen containers.)

These required server-side attributes are: `Wallet`, `CipherSuites`, and `Encryption`. See Create the ConfigMap for the server-side attributes for information on these attributes. Also see Server attributes for TLS in the *Oracle TimesTen In-Memory Database Security Guide*.

In addition to the `csWallet` and the `db.ini` metadata files, you may use other supported metadata files. See Configuration metadata details for information on these supported metadata files.

You can include these metadata files in one or more Kubernetes facilities (for example, in a Kubernetes Secret, in a ConfigMap, or in an init container). This ensures the metadata files are populated in the `/ttconfig` directory of the TimesTen containers. Note that there is no requirement as to how to get the metadata files into this `/ttconfig` directory. See Populating the /ttconfig directory.

The following example includes the `csWallet` metadata file in a Kubernetes Secret. It also creates the `db.ini`, the `adminUser`, and the `schema.sql` metadata files and includes these metadata files in a ConfigMap.

Create the Kubernetes Secret for the csWallet metadata file

This section creates the `cs-tls` Kubernetes Secret. The `cs-tls` Secret will contain the `csWallet` metadata file.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory. This example creates the `serverWallet` subdirectory. (The `serverWallet` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p serverWallet
   ```

2. Copy the `cwallet.sso` file into the `serverWallet` directory that you just created. Recall that the `cwallet.sso` file was generated when you used the `ttCreateCerts` utility to create the TLS certificates. Also recall that this file was located in the `/scratch/ttuser/instance_dir/instance1/conf/serverWallet` directory. See "Creating TLS certificates for replication and Client/Server" for information.

   ```
   % cp /scratch/ttuser/instance_dir/instance1/conf/serverWallet/cwallet.sso
   serverWallet/cwallet.sso
   ```

3. Create the Kubernetes Secret.

   In this example:

   - The name of the Secret is `cs-tls`. Replace `cs-tls` with a name of your choosing. (`cs-tls` is represented in **bold**.)
   - The name of the metadata file required for TLS for Client/Server is `csWallet` (represented in **bold**).
   - The location of the wallet directory is `serverWallet` (in this example, represented in **bold**). If you use a different directory, replace `serverWallet` with the name of your directory.
   - The name of the Oracle wallet: `cwallet.sso` (represented in **bold**).

   Use the `kubectl create` command to create the Secret:

   ```
   % kubectl create secret generic cs-tls
   --from-file=csWallet=serverWallet/cwallet.sso
   secret/cs-tls created
   ```

You have successfully created and deployed the `cs-tls` Kubernetes Secret. The `csWallet/cwallet.sso` file will later be available in the `/ttconfig` directory of the TimesTen containers. In addition, the file will be available in the `/tt/home/timesten/csWallet` directory of the TimesTen containers.

Create the ConfigMap for the server-side attributes

This section creates the `cs-tls` ConfigMap. This ConfigMap contains the `db.ini`, the `adminUser`, and the `schema.sql` metadata files.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_csTLS` subdirectory. (The `cm_csTLS` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p cm_csTLS
   ```

2. Navigate to the ConfigMap directory.

   ```
   % cd cm_csTLS
   ```

3. Create the `db.ini` file in this ConfigMap directory (`cm_csTLS`, in this example). In this `db.ini` file, define the server-side attributes for TLS for Client/Server. These server-side attributes will later be included in the `sys.odbc.ini file` located in the *`timesten_home`*`/conf` directory in your TimesTen containers. (Note that *`timesten_home`* is the TimesTen instance directory. This instance directory is `tt/home/timesten/instances/instance1` in your TimesTen containers.)

   These are the required server-side attributes for TLS for Client/Server:

   - `wallet`: This is the directory in your TimesTen containers that contains the server wallet. Specify `/tt/home/timesten/csWallet`.
   - `ciphersuites`: This is the cipher suite setting. Valid values are `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_256` or `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_384`, or both, comma separated and in order of preference. There is no default setting. For TLS to be used, the server and the client settings must include at least one common suite. This example specifies `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_256`. See Server attributes for TLS in the *Oracle TimesTen In-Memory Database Security Guide* for information on the cipher suite settings.
   - `encryption`: This is the encryption setting for the server. This example specifies the `required` setting. See Configuration on the server in the *Oracle TimesTen In-Memory Database Security Guide* for information on the valid encryption settings.

   This example also specifies the `PermSize` and the `DatabaseCharacterSet` connection attributes.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   wallet=/tt/home/timesten/csWallet
   ciphersuites=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
   encryption=required
   ```

4. Create the `adminUser` file in this ConfigMap directory (`cm_csTLS`, in this example). In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

```
vi adminUser

sampleuser/samplepw
```

5. Create the `schema.sql` file in this ConfigMap directory (`cm_csTLS`, in this example). In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator will automatically initialize your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

6. Create the ConfigMap. The files in the `cm_csTLS` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

   In this example:

   - The name of the ConfigMap is `cs-tls`. Replace `cs-tls` with a name of your choosing. (`cs-tls` is represented in **bold** in this example.)
   - This example uses `cm_csTLS` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_csTLS` with the name of your directory. (`cm_csTLS` is represented in **bold** in this example.)

   Use the `kubectl create` command to create the ConfigMap:

   ```
   % kubectl create configmap cs-tls --from-file=cm_csTLS
   configmap/cs-tls created
   ```

7. Use the `kubectl describe` command to verify the contents of the ConfigMap. (`cs-tls`, in this example.)

   ```
   % kubectl describe configmap cs-tls
   Name:         cs-tls
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>

   Data
   ====
   db.ini:
   ----
   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   wallet=/tt/home/timesten/csWallet
   ciphersuites=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
   encryption=required


   schema.sql:
   ----
   create sequence sampleuser.s;
   create table sampleuser.emp (id number not null primary key, name char (32));

   adminUser:
   ----
   sampleuser/samplepw
   ```

```
Events:  <none>
```

You have successfully created and deployed the `cs-tls` ConfigMap.

Create the TimesTenClassic object

This section creates the TimesTenClassic object. See "Defining and creating the TimesTenClassic object" and "About the TimesTenClassic object type" for detailed information on the TimesTenClassic object.

Perform these steps:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (In this example, `cstls`.) The YAML file contains the definitions for the TimesTenClassic object. See "TimesTenClassicSpecSpec" for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   In this example, the fields of particular interest for TLS Client/Server are:

   - `dbSecret`: This example uses one Kubernetes Secret (called `cs-tls`) for the `csWallet` metadata file.
   - `dbConfigMap`: This example uses one ConfigMap (called `cs-tls`). The `db.ini` file is contained in the `cs-tls` ConfigMap. Recall that the `db.ini` file contains the server-side attributes for TLS for Client/Server.

   In addition, this example includes:

   - `name`: Replace `cstls` with the name of your TimesTenClassic object.
   - `storageClassName`: Replace `oci` with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
   - `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. Note: This example assumes a production environment and uses a value of `250Gi` for `storageSize`. For demonstration purposes, a value of `50Gi` is adequate. See the `storageSize` and the `logStorageSize` entries in "Table 15-3" for information.
   - `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location and name of your image.
   - `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.

   ```
   % vi cstls.yaml

   apiVersion: timesten.oracle.com/v1
   kind: TimesTenClassic
   metadata:
     name: cstls
   spec:
     ttspec:
       storageClassName: oci
       storageSize: 250Gi
       image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
       imagePullSecret: sekret
       dbConfigMap:
       - cs-tls
   ```

```
    dbSecret:
    - cs-tls
```

2. Use the `kubectl create` command to create the TimesTenClassic object from the contents of the YAML file (in this example, `cstls.yaml`). Doing so begins the process of deploying your active standby pair of TimesTen databases in the Kubernetes cluster.

   ```
   % kubectl create -f cstls.yaml
   timestenclassic.timesten.oracle.com/cstls created
   ```

You have successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your TimesTen databases begins, but is not yet complete.

Monitor the deployment of the TimesTenClassic object

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the active standby pair as it is provisioned.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

   ```
   % kubectl get ttc cstls
   NAME    STATE          ACTIVE   AGE
   cstls   Initializing   None     15s
   ```

2. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

   ```
   % kubectl get ttc cstls
   NAME    STATE    ACTIVE    AGE
   cstls   Normal   cstls-0   3m30s
   ```

3. Use the `kubectl describe` command to view the active standby pair provisioning in detail.

   Note the following have been correctly set in the `cstls` TimesTenClassic object definition:

   - The `cs-tls` Secret has been correctly referenced in the `dbSecret` field (represented in **bold**).
   - The `cs-tls` Configmap has been correctly referenced in the `dbConfigMap` field (represented in **bold**).

   Note: Note all of the output is shown in this example.

   ```
   % kubectl describe ttc cstls
   Name:         cstls
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   API Version:  timesten.oracle.com/v1
   Kind:         TimesTenClassic
   Metadata:
     Creation Timestamp:  2021-10-17T19:08:03Z
     Generation:          1
     Resource Version:    75491472
     Self Link:
   /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/cstls
   ```

```
    UID:                    150128b3-10ac-11eb-b019-d681454a288b
  Spec:
    Ttspec:
      Db Config Map:
        cs-tls
      Db Secret:
        cs-tls
      Image:                container-registry.oracle.com/timesten/timesten:22.1.1.9.0
      Image Pull Policy:    Always
      Image Pull Secret:    sekret
      Storage Class Name:   oci
      Storage Size:         250Gi
...
Events:
  Type  Reason       Age    From        Message
  ----  ------       ----   ----        -------
  -     Create       4m21s  ttclassic   Service cstls created
  -     Create       4m21s  ttclassic   StatefulSet cstls created
  -     Create       4m21s  ttclassic   Secret tt150128b3-10ac-11eb-b019-
d681454a288b created
  -     StateChange  3m5s   ttclassic   Pod cstls-1 Daemon Up
  -     StateChange  3m5s   ttclassic   Pod cstls-0 Agent Up
  -     StateChange  3m5s   ttclassic   Pod cstls-0 Release 22.1.1.9.0
  -     StateChange  3m5s   ttclassic   Pod cstls-1 Agent Up
  -     StateChange  3m5s   ttclassic   Pod cstls-1 Release 22.1.1.9.0
  -     StateChange  3m5s   ttclassic   Pod cstls-0 Daemon Up
  -     StateChange  116s   ttclassic   Pod cstls-0 Database Loaded
  -     StateChange  116s   ttclassic   Pod cstls-0 Database Updatable
  -     StateChange  116s   ttclassic   Pod cstls-0 CacheAgent Not Running
  -     StateChange  116s   ttclassic   Pod cstls-0 RepAgent Not Running
  -     StateChange  116s   ttclassic   Pod cstls-0 RepState IDLE
  -     StateChange  116s   ttclassic   Pod cstls-0 RepScheme None
  -     StateChange  115s   ttclassic   Pod cstls-0 RepAgent Running
  -     StateChange  115s   ttclassic   Pod cstls-0 RepScheme Exists
  -     StateChange  115s   ttclassic   Pod cstls-0 RepState ACTIVE
  -     StateChange  96s    ttclassic   Pod cstls-1 Database Loaded
  -     StateChange  96s    ttclassic   Pod cstls-1 Database Not Updatable
  -     StateChange  96s    ttclassic   Pod cstls-1 CacheAgent Not Running
  -     StateChange  96s    ttclassic   Pod cstls-1 RepAgent Not Running
  -     StateChange  96s    ttclassic   Pod cstls-1 RepScheme Exists
  -     StateChange  96s    ttclassic   Pod cstls-1 RepState IDLE
  -     StateChange  90s    ttclassic   Pod cstls-1 RepAgent Running
  -     StateChange  84s    ttclassic   Pod cstls-1 RepState STANDBY
  -     StateChange  84s    ttclassic   TimesTenClassic was Initializing, now Normal
```

Your active standby pair of TimesTen databases are successfully deployed (as indicated by `Normal`.)

Configuration on the client

These sections cover the client requirements for TLS.

- Copy the client wallet
- Configure the client-side attributes

Copy the client wallet

When you used the `ttCreateCerts` utility to create TLS certificates, the `cwallet.sso` wallet file located in the `/scratch/ttuser/instance_dir/instance1/conf/ clientWallet` directory was generated. This file must be copied to the application container that is running your TimesTen client instance. See "Creating TLS certificates for replication and Client/ Server" for information on creating the TLS certificates.

This example uses the `kubectl cp` command to copy the `/scratch/ttuser/instance_dir/instance1/conf/clientWallet/cwallet.sso` file from your Linux development host to the application container running your TimesTen client instance.

1. Use the `kubectl exec -it` command to invoke the shell in the application container that contains your TimesTen client instance. (`cstls-0`, in this example).

   ```
   % kubectl exec -it cstls-0 -c tt -- /bin/bash
   ```

2. From the shell just invoked, and from the directory of your choice, create an empty subdirectory. This example creates the `clientWallet` subdirectory.

   ```
   % mkdir -p clientWallet
   ```

3. From your Linux development host, use the `kubectl cp` command to copy the `cwallet.sso` file from the `/scratch/ttuser/instance_dir/instance1/conf/clientWallet` directory on your Linux development host to the `clientWallet` directory that you just created. (This directory is located in the application container that is running your TimesTen client instance.) Recall that the `cwallet.sso` file was generated when you used the `ttCreateCerts` utility to create the TLS certificates. See Creating TLS certificates for replication and Client/Server for information.

   ```
   % kubectl cp /scratch/ttuser/instance_dir/instance1/conf/clientWallet/
   cwallet.sso cstls-0:clientWallet/cwallet.sso -c tt
   ```

4. From your shell, verify the `cwallet.sso` file is located in the `clientWallet` directory.

   ```
   % ls clientWallet
   cwallet.sso
   ```

You have successfully copied the `cwallet.sso` client wallet file to the application container that is running your TimesTen client instance.

Configure the client-side attributes

You must set client-side attributes for TLS for Client/Server. The attributes can be set in the client DSN definition in *`timesten_home`*`/conf/sys.odbc.ini` or in an appropriate Client/Server connection string. See Using Client/Server drivers for additional information.

These are the required client-side attributes for TLS for Client/Server:

- `wallet`: This is the directory that contains the `cwallet.sso` client wallet file. This directory is located in your application container that is running the TimesTen client instance. There is no default directory. In this example, recall that the `clientWallet` directory was created to denote this directory. (See Copy the client wallet for information.) For purposes of this example, the full path to the `clientWallet` directory is `/tt/home/timesten/clientWallet`. Therefore, in this example, `/tt/home/timesten/clientWallet` is used to denote this directory.
- `ciphersuites`: This is the cipher suite setting. Valid values are `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_256` or `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_384`, or both, comma separated and in order of preference. There is no default setting. For TLS to be used, the server and the

client settings must include at least one common suite. This example specifies `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_256`. See Configuration for TLS for Client/Server in the *Oracle TimesTen In-Memory Database Security Guide* for information on the cipher suite settings.

- `encryption`: This is the encryption setting for the client. This example specifies the `required` setting. See Configuration for TLS for Client/Server in the *Oracle TimesTen In-Memory Database Security Guide* for information on the valid encryption settings.

This example uses a connection string to connect to the `cstsl` database as the `sampleuser` user. The `sampleuser` user was created by the Operator and already exists in the `cstsl` database. The example then uses the `sqlgetconnectattr` command from `ttIsqlCS` on the client to verify TLS is configured correctly on the Server and on the Client and TLS is being used.

1. Connect to the database.

```
% ttIsqlcs -connstr "TTC_SERVER1=cstls-0.cstls.mynamespace.svc.cluster.local;
TTC_SERVER2=cstls-1.cstls.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=cstls;UID=sampleuser;PWD=samplepw;
WALLET=tt/home/timesten/clientWallet;
CIPHERSUITES=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
ENCRYPTION=required";

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "TTC_SERVER1=cstls-0.cstls.mynamespace.svc.cluster.local;
TTC_SERVER2=cstls-1.cstls.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=cstls;UID=sampleuser;PWD=********;
WALLET=tt/home/timesten/clientWallet;
CipherSuites=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
ENCRYPTION=REQUIRED;";
Connection successful:
DSN=;TTC_SERVER=cstls-0.cstls.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=cstls;UID=sampleuser;
DATASTORE=/tt/home/timesten/datastore/
cstls;DATABASECHARACTERSET=AL32UTF8;AUTOCREATE=0;PERMSIZE=200;
DDLREPLICATIONLEVEL=3;FORCEDISCONNECTENABLED=1;(SERVER)ENCRYPTION=Required;
(SERVER)WALLET=file:/tt/home/timesten/csWallet;(client)Encryption=Required;
(client)Wallet=/tt/home/timesten/clientWallet;
(client)CipherSuites=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
(Default setting AutoCommit=1)
```

2. Use the `sqlgetconnectattr` command in `ttIsqlCS` to verify TLS is being used. A return value of `1` indicates TLS is being used.

```
Command> sqlgetconnectattr tt_tls_session;
TT_TLS_SESSION = 1 (SQL_TRUE)
```

You have successfully connected to the database and verified that TLS for Client/Server is being used.

13 Handling Failover and Recovery in TimesTen Classic

This chapter is specific to active standby pair of Classic TimesTen databases. It illustrates how the TimesTen Operator recovers from failure.

- Handling failover and recovery
- An example illustrating the failover and recovery process

Handling failover and recovery

The Operator automatically detects failures of the active TimesTen database and the standby TimesTen database and works to fix any failures. When the Operator detects a failure of the active database, it promotes the standby TimesTen database to be the active. Client/server applications that are using the database are automatically reconnected to the new active database. Transactions in flight are rolled back. Prepared statements need to be re-prepared by the applications. The Operator will configure a new standby database.

An example illustrating the failover and recovery process

This example simulates a failure of the active TimesTen database. This is for demonstration purposes only. Do not do this in a production environment.

1. Use the `kubectl delete pod` command to delete the active database (`sample-0` in this case)

   ```
   % kubectl delete pod sample-0
   ```

2. Use the `kubectl describe` command to observe how the Operator recovers from the failure. The Operator promotes the standby database (`sample-1`) to be active. Any applications that were connected to the `sample-0` database are automatically reconnected to the `sample-1` database by TimesTen. After a brief outage, the applications can continue to use the database. See "Monitoring the health of the active standby pair of databases" for information on the health and states of the active standby pair.

   Note: In this example, the text for the `Message` column displays on two lines for three state changes. However, the actual output displays on one line for each of these three state changes.

   ```
   % kubectl describe ttc sample
   Name:         sample
   ...
   Events:
     Type  Reason       Age   From       Message
     ----  ------       ----  ----       -------
     -     StateChange  2m1s  ttclassic  TimesTenClassic sample: was Normal, now
   ActiveDown
     -     StateChange  115s  ttclassic  Pod sample-1 Database Updatable: Yes
     -     StateChange  115s  ttclassic  TimesTenClassic sample:was ActiveDown, now
   StandbyDown
   ```

```
  -     StateChange  115s   ttclassic  Pod sample-1 RepState ACTIVE
  -     StateChange  110s   ttclassic  Pod sample-0 High Level State Unknown
  -     StateChange  63s    ttclassic  Pod sample-0 Pod Phase Running
  -     StateChange  63s    ttclassic  Pod sample-0 Agent Up
  -     StateChange  63s    ttclassic  Pod sample-0 Instance Exists
  -     StateChange  63s    ttclassic  Pod sample-0 Daemon Up
  -     StateChange  63s    ttclassic  Pod sample-0 Database None
  -     StateChange  42s    ttclassic  Pod sample-0 Database Loaded
  -     StateChange  42s    ttclassic  Pod sample-0 Database Updatable: No
  -     StateChange  42s    ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  42s    ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  42s    ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  42s    ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  36s    ttclassic  Pod sample-0 High Level State Healthy
  -     StateChange  36s    ttclassic  Pod sample-0 RepState STANDBY
  -     StateChange  36s    ttclassic  TimesTenClassic sample:was
StandbyDown,now Normal
```

Kubernetes has automatically respawned a new `sample-0` Pod to replace the Pod you deleted. The Operator configured TimesTen within that Pod, bringing the database in the Pod up as the new standby database. The replicated pair of databases are once again functioning normally.

14 Performing Upgrades

This chapter shows you how to upgrade the TimesTen Operator. It also shows you how to upgrade TimesTen Classic to a new patch or patchset. The process also applies to downgrades.

For information on TimesTen releases and patches, see Overview of release numbers in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Topics include:

About new TimesTen container images

The TimesTen Kubernetes Operator lets you upgrade the TimesTen Operator and TimesTen Classic to a new patch or patch set release. There are separate procedures in place for upgrading the Operator and for upgrading TimesTen Classic.

Both the Operator as well as each TimesTenClassic and TimesTenScaleout object require a TimesTen container image. You tell the Operator the location of the image registry and the name of the container image when you initially deploy the Operator and a TimesTenClassic or a TimesTenScaleout object.

When you upgrade the Operator to a new patch or patch set, you need to decide if you want to use a TimesTen container image that is located in a `timesten` repository on `container-registry.oracle.com` or if you want to build your own TimesTen container image. It is your choice as to which option you choose. For information about TimesTen container images, see About TimesTen container images.

Option 1: Use `container-registry.oracle.com/timesten` or `container-registry.oracle.com/timesten-xe`.

There are multiple container images available on the `container-registry.oracle.com/timesten` and `container-registry.oracle.com/timesten-xe` repositories. Browse the repositories and choose a container image that you want to use for the upgrade.

For example, if you want to upgrade to TimesTen release `22.1.1.9.0`, a suitable TimesTen container image is:

```
container-registry.oracle.com/timesten/timesten:22.1.1.9.0
```

After you choose the container image, you must obtain the Operator manifest files from it.

For information about how to use a TimesTen container image located in a `container-registry.oracle.com` `timesten` repository, and how to obtain the Operator manifest files from it, see Option 1: Use the official TimesTen container images.

Option 2: Build a TimesTen container image

The TimesTen distribution contains the TimesTen Operator distribution. The TimesTen Operator distribution provides a Dockerfile for building a TimesTen container image. For the upgrade, choose the pertinent TimesTen patch or patchset distribution. Download and unpack it. Next, unpack the Operator distribution. The Operator distribution not only includes the Dockerfile to build your container image, but also contains the Operator manifest files.

For example, if you want to upgrade to TimesTen release `22.1.1.9.0`, a suitable TimesTen distribution is:

```
timesten221190.server.linux8664.zip
```

For information about building a TimesTen container image, see Option 2: Build the TimesTen container image.

After you complete the tasks for either Option 1 or Option 2, you should know the following:

- Name of image registry

  For example, `container-registry.oracle.com/timesten` or `phx.ocir.io/`*`youraccount`*

- Name of image

  For example, `timesten:22.1.1.9.0`

- Directory that contains the Operator manifest files

  For example, `new_kube_files`

  The Operator manifest files include the new `crd.yaml` and the new `service_account.yaml` files. You update your Kubernetes cluster with these files. There is also a new `operator.yaml` file. You edit this file with the location and name of the new TimesTen container image. These tasks are described in Upgrade the Operator.

For the examples in the upcoming sections, let's assume:

- Location of the image registry is `container-registry.oracle.com/timesten`.
- Name of the image is `timesten:22.1.1.9.0`.
- Operator manifest files reside in the `new_kube_files` directory on your development host.

Upgrade the Operator

The tasks in this section show you how to upgrade the Operator to a new patch or patchset. You can perform an Operator upgrade while there are TimesTenClassic and TimesTenScaleout objects deployed in your namespace.

Let's first update the Kubernetes cluster with the new CRD and service account.

1. On your development host, change to the `new_kube_files` directory. Next, replace the CRD running in your Kubernetes cluster.

   ```
   cd new_kube_files
   ```

   ```
   kubectl replace -f crd.yaml
   ```

   The output is the following.

   ```
   customresourcedefinition.apiextensions.k8s.io/
   timestenclassics.timesten.oracle.com/
   timestenscaleouts.timesten.oracle.com replaced
   ```

2. Replace the service account.

   ```
   kubectl replace -f service_account.yaml
   ```

   The output is the following.

   ```
   role.rbac.authorization.k8s.io/timesten-operator replaced
   serviceaccount/timesten-operator replaced
   rolebinding.rbac.authorization.k8s.io/timesten-operator replaced
   ```

   Next, let's upgrade the Operator.

3. (Optional): Confirm there is an Operator running.

   ```
   kubectl get pods
   ```

   The output is similar to the following.

   ```
   NAME                                 READY   STATUS    RESTARTS        AGE
   sample-0                             2/2     Running   0               14h
   sample-1                             2/2     Running   0               14h
   sample2-0                            2/2     Running   0               14h
   sample2-1                            2/2     Running   0               14h
   timesten-operator-778878dc6b-4mc77   1/1     Running   0               15h
   ```

   The name of the Operator is `timesten-operator-778878dc6b-4mc77`. The other Pods are associated with `sample` and `sample2` TimesTenClassic objects in your namespace.

4. Upgrade the `timesten-operator` Deployment.

Edit these fields in the `operator.yaml` file:

- `replicas: 1`

  Replace `1` with the number of copies of the Operator that you want to run. A value of `1` is acceptable for development and testing. However, you can run more than one replica for high availability purposes.
- Replace `sekret` with the name of the image pull secret that you want Kubernetes to use to pull images from your registry.
- Replace the `image` line to reference the new TimesTen container image. In this example, the new image is `container-registry.oracle.com/timesten/timesten:22.1.1.9.0`.

```
vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timesten-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timesten-operator
  template:
    metadata:
      labels:
        name: timesten-operator
    spec:
      serviceAccountName: timesten-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timesten-operator
          image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
          command:
          - /timesten/operator/operator/timesten-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "timesten-operator"
          securityContext:
              runAsNonRoot: true
              privileged: false
              allowPrivilegeEscalation: false
              capabilities:
                  drop:
                    - all
```

5. Update the `timesten-operator` Deployment.

   ```
   kubectl replace -f operator.yaml
   ```

   The output is the following.

   ```
   deployment.apps/timesten-operator replaced
   ```

6. Verify that the new Operator is running.

   ```
   kubectl get pods
   ```

   The output is similar to the following.

   ```
   NAME                                  READY   STATUS    RESTARTS     AGE
   sample-0                              2/2     Running   0            15h
   sample-1                              2/2     Running   0            15h
   sample2-0                             2/2     Running   0            15h
   sample2-1                             2/2     Running   0            15h
   timesten-operator-6f9d96bdfc-h22lm    1/1     Running   0            9s
   ```

   The name of the new Operator is `timesten-operator-6f9d96bdfc-h22lm`.

7. Review the new `timesten-operator` Deployment.

   ```
   kubectl describe deployment timesten-operator
   ```

   The output is similar to the following.

   ```
   Name:                   timesten-operator
   Namespace:              mynamespace
   CreationTimestamp:      Sun, 08 Jan 2023 01:22:28 +0000
   Labels:                 <none>
   Annotations:            deployment.kubernetes.io/revision: 3
   Selector:               name=timesten-operator
   Replicas:               1 desired | 1 updated | 1 total | 1 available | 0
   unavailable
   StrategyType:           RollingUpdate
   MinReadySeconds:        0
   RollingUpdateStrategy:  25% max unavailable, 25% max surge
   Pod Template:
     Labels:           name=timesten-operator
     Service Account:  timesten-operator
     Containers:
      timesten-operator:
       Image:      container-registry.oracle.com/timesten/timesten:22.1.1.9.0
       Port:       <none>
       Host Port:  <none>
   ```

```
      Command:
        /timesten/operator/operator/timesten-operator
      Environment:
        WATCH_NAMESPACE:   (v1:metadata.namespace)
        POD_NAME:          (v1:metadata.name)
        OPERATOR_NAME:    timesten-operator
      Mounts:             <none>
    Volumes:              <none>
  Conditions:
    Type           Status  Reason
    ----           ------  ------
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
  NewReplicaSet:   timesten-operator-6f9d96bdfc (1/1 replicas created)
  Events:
    ...
```

You have successfully updated the `timesten-operator` Deployment. The new Operator is using the `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` container image, and is automatically managing any existing TimesTenClassic and TimesTenScaleout objects in your namespace.

About upgrading TimesTen Classic

The TimesTen Kubernetes Operator supports the upgrade of TimesTen Classic to a new patch or patchset.

There are two options for upgrading:

- `Auto` (default): The Operator does the upgrade for you.
- `Manual`: You do the upgrade manually.

The `.spec.ttspec.imageUpgradeStrategy` datum for a TimesTenClassic object lets you choose the type of upgrade. You set a value for this datum when you initially create and deploy a TimesTenClassic object. For more information about the `imageUpgradeStrategy` datum, see TimesTenClassicSpecSpec.

No matter what type of upgrade you choose, what happens during the upgrade is similar. The standby is terminated first. It takes some time for the standby to come back up. During this wait period, the standby is upgraded to the new release. During this upgrade of the standby, depending on your replication configuration, there may be disruption on the active database. This may impact your applications. Next, the failover from the active to the standby occurs. The active is terminated. There is a wait period for the former active to come back up. During this wait period, the active is upgraded to the new release. The standby database is promoted to be the active and the former active becomes the standby.

There are examples illustrating both types of upgrades later in Perform an automated upgrade and Perform a manual upgrade.

Note:

If you are using AWT cache groups, the standby is normally responsible for transmitting committed transactions from TimesTen to the Oracle Database. While the standby is being upgraded, the active takes on this responsibility. This may increase the load on the active. In addition, part of the upgrade process involves copying the database from the active to the standby. This also increases the workload on the active. These increases may temporarily reduce the performance of the active database.

Ensure you perform an upgrade at the appropriate time. TimesTen recommends that you do not perform upgrades at the busiest time of a production day. Applications see shortages and perhaps reduced performance as a result of the upgrade procedure.

Perform an automated upgrade

The `.spec.ttspec.imageUpgradeStrategy` datum of a TimesTenClassic object determines the upgrade strategy. You can use the `kubectl get ttc -o yaml` command to determine the value of this datum for any TimesTenClassic object. For example, let's assume there is a `sample` TimesTenClassic object in our namespace. Let's review the setting for the `.spec.ttspec.imageUpgradeStrategy` datum for `sample`.

```
kubectl get ttc sample -o yaml
```

The output is similar to the following. Note: Not all output is shown.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
    ...
  name: sample
  namespace: mynamespace
  resourceVersion: "74826407"
  uid: 76038b9b-4635-4974-88cf-385739499ec4
spec:
  ...
  ttspec:
    additionalMemoryRequest: 1Gi
    automaticMemoryRequests: true
    daemonLogCPURequest: 200m
    daemonLogMemoryRequest: 20Mi
    dbConfigMap:
    - sample
    exporterCPURequest: 200m
    exporterMemoryRequest: 200Mi
    ...
    imageUpgradeStrategy: Auto
    memoryWarningPercent: 90
...
```

The value for `.spec.ttspec.imageUpgradeStrategy` is `Auto`.

Let's walk through the steps for an automated upgrade.

Note:

Recall that when you do an automated upgrade, your databases are taken down, restarted, and failed over **immediately**. Do not perform this procedure at the busiest time of your production day. Applications see short outages and perhaps reduced performance as a result of the upgrade procedure.

Modify the TimesTenClassic object: automated upgrade

The automated upgrade process requires you to modify the `.spec.ttspec.image` datum of a TimesTenClassic object to reference the new TimesTen container image. After you modify the TimesTenClassic object to reference the new TimesTen image, the Operator notices the change and modifies the StatefulSet that it created. The Operator then starts the upgrade process. You can use the `kubectl describe` command to monitor this upgrade process.

1. On your development host, edit the `sample` TimesTenClassic object, changing the `.spec.ttspec.image` datum to reference the new TimesTen container image. In this example, the location and name of the new container image is `container-registry.oracle.com/timesten/timesten:22.1.1.9.0`.

   Note: Not all output is shown.

```
kubectl edit timestenclassic sample

# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while
saving this file will be
# reopened with the relevant failures.
#
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  ...
  name: sample
  namespace: mynamespace
  resourceVersion: "74909603"
  uid: 76038b9b-4635-4974-88cf-385739499ec4
spec:
...
  ttspec:
    additionalMemoryRequest: 1Gi
    automaticMemoryRequests: true
    daemonLogCPURequest: 200m
    daemonLogMemoryRequest: 20Mi
    dbConfigMap:
    - sample
    exporterCPURequest: 200m
    exporterMemoryRequest: 200Mi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullPolicy: Always
    imagePullSecret: sekret
    imageUpgradeStrategy: Auto
    memoryWarningPercent: 90
    storageClassName: oci
    storageSize: 250G
```

The output is the following.

```
timestenclassic.timesten.oracle.com/sample edited
```

2. Verify that the Operator has modified the `sample` StatefulSet and replaced the image with the new image.

```
kubectl describe statefulset sample
```

The output is similar to the following.

```
Name:                   sample
Namespace:              mynamespace
...
Replicas:               2 desired | 2 total
Update Strategy:        OnDelete
Pods Status:            1 Running / 1 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       app=sample
                database.timesten.oracle.com=sample
  Annotations:  TTC: 76038b9b-4635-4974-88cf-385739499ec4
  Init Containers:
   ttinit:
    Image:       container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
    Environment:
      TT_OPERATOR_MANAGED:      1
      TIMESTEN_HOME:            /tt/home/timesten/instances/instance1
      LD_LIBRARY_PATH:          /tt/home/timesten/instances/instance1/
ttclasses/lib:/tt/home/timesten/instances/instance1/install/lib:/tt/home/
timesten/instances/instance1/install/ttoracle_home/instantclient_11_2:/tt/
home/timesten/instances/instance1/install/ttoracle_home/instantclient
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
      TT_INIT_CONTAINER:        1
      TTC_UID:                  76038b9b-4635-4974-88cf-385739499ec4
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
  Containers:
   tt:
    Image:        container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Ports:        8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:   0/TCP, 0/TCP, 0/TCP, 0/TCP
    Environment:
      TT_OPERATOR_MANAGED:      1
      TIMESTEN_HOME:            /tt/home/timesten/instances/
instance1
      LD_LIBRARY_PATH:          /tt/home/timesten/instances/
instance1/ttclasses/lib:/tt/home/timesten/instances/instance1/
install/lib:/tt/home/timesten/instances/instance1/install/
ttoracle_home/instantclient_11_2:/tt/home/timesten/instances/
instance1/install/ttoracle_home/instantclient
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
   daemonlog:
    Image:       container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Port:        <none>
    Host Port:   <none>
    Command:
      sh
      -c
      /bin/bash <<'EOF'
      while [ 1 ] ; do tail --follow=name /tt/home/timesten/
instances/instance1/diag/ttmesg.log --max-unchanged-stats=5; sleep
1; done
      exit 0
      EOF
    Requests:
      cpu:      100m
      memory:   20Mi
    Environment:
      TIMESTEN_HOME:        /tt/home/timesten/instances/instance1
      TT_OPERATOR_MANAGED:  1
      LD_LIBRARY_PATH:      /tt/home/timesten/instances/instance1/
ttclasses/lib:/tt/home/timesten/instances/instance1/install/lib:/tt/
home/timesten/instances/instance1/install/ttoracle_home/
instantclient_11_2:/tt/home/timesten/instances/instance1/install/
ttoracle_home/instantclient
    Mounts:
      /tt from tt-persistent (rw)
  Volumes:
   tt-agent:
    Type:         Secret (a volume populated by a Secret)
    SecretName:   tt76038b9b-4635-4974-88cf-385739499ec4
    Optional:     false
   tt-config:
    Type:                Projected (a volume that contains injected
data from multiple sources)
    ConfigMapName:       sample
    ConfigMapOptional:   <nil>
```

```
Volume Claims:
  Name:          tt-persistent
  StorageClass:  oci
  Labels:        <none>
  Annotations:   <none>
  Capacity:      50G
  Access Modes:  [ReadWriteOnce]
Events:
  Type    Reason            Age                 From
Message
  ----    ------            ----                ----
-------
  Normal  SuccessfulCreate  22s (x2 over 18h)  statefulset-controller
create Pod sample-1 in StatefulSet sample successful
```

You have modified the `sample` TimesTenClassic object to use the new TimesTen container image. You are now ready to monitor the automated upgrade process performed by the Operator.

Monitor the automated upgrade

You can monitor the automated upgrade process performed by the Operator. These steps are optional.

1. Use the `kubectl get` command to assess the state of the `sample` TimesTenClassic object.

   ```
   kubectl get ttc sample
   ```

   Note that the initial state is `StandbyDown`.

   ```
   NAME      STATE         ACTIVE     AGE
   sample    StandbyDown   sample-0   18h
   ```

   Wait a few minutes, then run the command again.

   ```
   kubectl get ttc sample
   ```

   Note that the state has changed to `Normal`.

   ```
   NAME      STATE     ACTIVE     AGE
   sample    Normal    sample-1   18h
   ```

   The Operator promoted `sample-1` to be the active.

2. Use the `kubectl describe` command to observe how the Operator promoted the standby database (`sample-1`) to be the active.

   ```
   kubectl describe ttc sample
   ```

The output is similar to the following.

```
Name:         sample
Namespace:    mynamespace

Kind:         TimesTenClassic
Metadata:
    ...
Spec:
  ...
  Ttspec:
    Additional Memory Request:  1Gi
    Automatic Memory Requests:  true
    Daemon Log CPU Request:     200m
    Daemon Log Memory Request:  20Mi
    Db Config Map:
      sample
    Exporter CPU Request:     200m
    Exporter Memory Request:  200Mi
    Image:                    container-registry.oracle.com/
timesten/timesten:22.1.1.9.0
    Image Pull Policy:        Always
    Image Pull Secret:        sekret
    Image Upgrade Strategy:   Auto
    Memory Warning Percent:   90
    Storage Class Name:       oci
    Storage Size:             250G
Status:
  ...
  Pod Status:
    Active:           false
    Admin User File:  true
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Cache User File:      false
    Cg File:              false
    Db Status:
      Db:   Loaded
      ...
    Disable Return:                                         false
    Has Been Seen:                                          true
    High Level State:                                       Healthy
    Initialized:                                            true
    Intended State:                                         Standby
    Last High Level State Switch:                           1673208466
    Local Commit:                                           false
    Name:                                                   sample-0
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1673208466
      Pod IP:               192.0.2.1
      Pod Phase:            Running
    Prev Active:            false
    Prev High Level State:  Healthy
    Prev Image:             container-registry.oracle.com/timesten/
timesten:22.1.1.7.0
    Prev Intended State:    Active
    Prev Ready:             true
    Ready:                  true
    Replication Status:
      Last Time Rep State Changed:  1673208234
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Scaleout Status:
      Instance Type:  classic
    Schema File:      true
    Timesten Status:
      Daemon:         Up
      Instance:       Exists
      Release:        22.1.1.9.0
    Tt Pod Type:      Database
    Using Twosafe:    false
    Active:           true
    Admin User File:  true
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Cache User File:      false
    Cg File:              false
    Db Status:
      Db:  Loaded
      ...
    Name:                   sample-1
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1673208466
      Pod IP:               192.0.2.2
      Pod Phase:            Running
    Prev Active:            true
    Prev High Level State:  Healthy
    Prev Image:             container-registry.oracle.com/timesten/
timesten:22.1.1.7.0
    Prev Intended State:    Standby
    Prev Ready:             true
    Ready:                  true
    Replication Status:
      Last Time Rep State Changed:  1673208191
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    ACTIVE
    Scaleout Status:
      Instance Type:  classic
    Schema File:      true
    Timesten Status:
      Daemon:             Up
      Instance:           Exists
      Release:            22.1.1.9.0
    Tt Pod Type:          Database
    Using Twosafe:        false
  Prev High Level State:  StandbyDown
  Prev Reexamine:
  Prev Stop Managing:
  Rep Create Statement:   create active standby pair "sample" on
"sample-0.sample.mynamespace.svc.cluster.local", "sample" on
"sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store
"sample" on "sample-0.sample.mynamespace.svc.cluster.local" PORT
4444 FAILTHRESHOLD 0 store "sample" on
"sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD 0
  Rep Port:               4444
  Rep Start Fail Count:   0
  Standby Cache Agent:    Not Running
  Standby Down Standby AS:
    Async Id:           5f083b21-20a9-4b93-93bf-8a42a160a0bc
    Destroy Db:         true
    Id:                 8bd89bbd-fb69-4249-822d-4dd729443e4b
    Pod Name:           sample-0
    Rep Duplicate:      true
    Start Rep Agent:    true
    Status:             complete
  Standby Perm In Use:  17134
  Standby Perm Size:    200
  Standby Rep Agent:    Running
  Status Version:       1.0
  Using Twosafe:        false
Events:
  Type     Reason       Age    From      Message
  ----     ------       ----   ----      -------
  Normal   Upgrade      6m41s  timesten  Image updated, automatic
upgrade started
  Normal   Upgrade      6m41s  timesten  Deleted standby pod
sample-1 during upgrade
  Normal   Info         6m29s  timesten  Pod sample-1 Agent Down
  Normal   StateChange  6m29s  timesten  Pod sample-1 is Not Ready
  Warning  StateChange  6m29s  timesten  TimesTenClassic was
Normal, now ActiveTakeover
  Normal   StateChange  6m23s  timesten  TimesTenClassic was
ActiveTakeover, now StandbyDown
  Normal   Info         5m4s   timesten  Pod sample-1 Agent Up
  Normal   Info         5m4s   timesten  Pod sample-1 Instance
Exists
  Normal   Info         5m4s   timesten  Pod sample-1 Daemon Down
  Normal   Info         5m3s   timesten  Pod sample-1 Daemon Up
  Normal   Info         5m3s   timesten  Pod sample-1 Database
Unloaded
  Normal   Info         5m1s   timesten  Pod sample-1 Database None
  Normal   StateChange  4m46s  timesten  Pod sample-1 RepState IDLE
  Normal   Info         4m46s  timesten  Pod sample-1 Database
Loaded
```

```
  Normal   Info         4m46s  timesten  Pod sample-1 RepScheme Exists
  Normal   Info         4m46s  timesten  Pod sample-1 RepAgent Not Running
  Normal   Info         4m40s  timesten  Pod sample-1 RepAgent Running
  Normal   StateChange  4m40s  timesten  TimesTenClassic was StandbyDown,
now StandbyStarting
  Normal   StateChange  4m40s  timesten  TimesTenClassic was
StandbyStarting, now StandbyCatchup
  Normal   StateChange  4m35s  timesten  Pod sample-1 RepState STANDBY
  Normal   StateChange  4m34s  timesten  TimesTenClassic was
StandbyCatchup, now Normal
  Normal   Upgrade      4m3s   timesten  Deleted active pod sample-0
during upgrade
  Normal   Info         3m52s  timesten  Pod sample-0 Agent Down
  Normal   StateChange  3m52s  timesten  Pod sample-0 is Not Ready
  Normal   StateChange  3m52s  timesten  Pod sample-0 is Not Active Ready
  Warning  StateChange  3m52s  timesten  TimesTenClassic was Normal, now
ActiveDown
  Normal   StateChange  3m52s  timesten  Pod sample-1 is Ready
  Normal   Info         3m50s  timesten  Pod sample-1 Database Updatable
  Normal   StateChange  3m50s  timesten  Pod sample-1 RepState ACTIVE
  Normal   StateChange  3m50s  timesten  Pod sample-1 is Not Ready
  Normal   StateChange  3m50s  timesten  TimesTenClassic was ActiveDown,
now ActiveTakeover
  Normal   StateChange  3m45s  timesten  Pod sample-1 is Ready
  Normal   StateChange  3m45s  timesten  Pod sample-1 is Active Ready
  Normal   StateChange  3m45s  timesten  TimesTenClassic was
ActiveTakeover, now StandbyDown
  Normal   Info         3m1s   timesten  Pod sample-0 Agent Up
  Normal   Info         3m1s   timesten  Pod sample-0 Instance Exists
  Normal   Info         3m1s   timesten  Pod sample-0 Daemon Down
  Normal   Info         3m1s   timesten  Pod sample-0 Daemon Up
  Normal   Info         3m1s   timesten  Pod sample-0 Database Unloaded
  Normal   Info         2m58s  timesten  Pod sample-0 Database None
  Normal   Info         2m43s  timesten  Pod sample-0 Database Loaded
  Normal   Info         2m43s  timesten  Pod sample-0 RepAgent Not Running
  Normal   Info         2m43s  timesten  Pod sample-0 RepScheme Exists
  Normal   StateChange  2m43s  timesten  Pod sample-0 RepState IDLE
  Normal   Info         2m37s  timesten  Pod sample-0 RepAgent Running
  Normal   StateChange  2m37s  timesten  Pod sample-0 RepState STANDBY
  Normal   StateChange  2m37s  timesten  Pod sample-0 is Ready
  Normal   Upgrade      2m37s  timesten  Upgrade of active complete
  Normal   Upgrade      2m37s  timesten  Upgrade completed in 244 secs
  Normal   StateChange  2m37s  timesten  TimesTenClassic was StandbyDown,
now Normal
```

The automated upgrade is successful. The active and standby Pods are running the new TimesTen image, which contains the new TimesTen release.

Perform a manual upgrade

This section describes the process for performing a manual upgrade of your TimesTenClassic objects and includes the following subsections:

- Modify the TimesTenClassic object: manual upgrade

- Upgrade the standby database
- Fail over

Modify the TimesTenClassic object: manual upgrade

The `manual` upgrade process requires you to modify the `.spec.ttspec.image` datum of a TimesTenClassic object to reference the new TimesTen container image. After you modify the TimesTenClassic object to reference the new TimesTen image, the Operator notices the change and modifies the StatefulSet that it created.

Let's modify the TimesTenClassic `sample2` object to reference the new TimesTen image. Let's assume the `sample2` object's `.spec.ttspec.imageUpgradeStrategy` is `Manual`.

1. On your development host, edit the `sample` TimesTenClassic object, changing the `.spec.ttspec.image` datum to reference the new TimesTen container image. In this example, the location and name of the new container image is `container-registry.oracle.com/timesten/timesten:22.1.1.9.0`.

   Note: Not all output is shown.

```
kubectl edit timestenclassic sample2

# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while
saving this file will be
# reopened with the relevant failures.
#
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  ...
  name: sample2
  namespace: mynamespace
  resourceVersion: "74928067"
  uid: e6d49f75-05ed-4b4b-9412-4a577ad19bbe
spec:
...
  ttspec:
    additionalMemoryRequest: 1Gi
    automaticMemoryRequests: true
    daemonLogCPURequest: 200m
    daemonLogMemoryRequest: 20Mi
    dbConfigMap:
    - sample2
    exporterCPURequest: 200m
    exporterMemoryRequest: 200Mi
    image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    imagePullPolicy: Always
    imagePullSecret: sekret
    imageUpgradeStrategy: Manual
    memoryWarningPercent: 90
    storageClassName: oci
    storageSize: 250G
...
```

The output is the following.

```
timestenclassic.timesten.oracle.com/sample2 edited
```

2. Verify that the Operator has modified the `sample2` StatefulSet and replaced the image with the new image.

   ```
   kubectl describe statefulset sample2
   ```

   The output is similar to the following.

   ```
   Name:               sample2
   Namespace:          mynamespace
   ...
   Replicas:           2 desired | 2 total
   Update Strategy:    OnDelete
   Pods Status:        2 Running / 0 Waiting / 0 Succeeded / 0 Failed
   Pod Template:
     Labels:       app=sample2
                   database.timesten.oracle.com=sample2
     Annotations:  TTC: e6d49f75-05ed-4b4b-9412-4a577ad19bbe
     Init Containers:
      ttinit:
       Image:       container-registry.oracle.com/timesten/
   timesten:22.1.1.9.0
       Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
       Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
       Environment:
         TT_OPERATOR_MANAGED:      1
         TIMESTEN_HOME:            /tt/home/timesten/instances/instance1
         LD_LIBRARY_PATH:          /tt/home/timesten/instances/instance1/
   ttclasses/lib:/tt/home/timesten/instances/instance1/install/lib:/tt/home/
   timesten/instances/instance1/install/ttoracle_home/instantclient_11_2:/tt/
   home/timesten/instances/instance1/install/ttoracle_home/instantclient
         TT_REPLICATION_TOPOLOGY:  activeStandbyPair
         TT_INIT_CONTAINER:        1
         TTC_UID:                  e6d49f75-05ed-4b4b-9412-4a577ad19bbe
       Mounts:
         /tt from tt-persistent (rw)
         /ttagent from tt-agent (rw)
         /ttconfig from tt-config (rw)
     Containers:
      tt:
       Image:       container-registry.oracle.com/timesten/
   timesten:22.1.1.9.0
       Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
       Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
       Environment:
         TT_OPERATOR_MANAGED:      1
         TIMESTEN_HOME:            /tt/home/timesten/instances/instance1
   ```

```
      LD_LIBRARY_PATH:          /tt/home/timesten/instances/
instance1/ttclasses/lib:/tt/home/timesten/instances/instance1/
install/lib:/tt/home/timesten/instances/instance1/install/
ttoracle_home/instantclient_11_2:/tt/home/timesten/instances/
instance1/install/ttoracle_home/instantclient
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
   daemonlog:
    Image:      container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Port:       <none>
    Host Port:  <none>
    Command:
      sh
      -c
      /bin/bash <<'EOF'
      while [ 1 ] ; do tail --follow=name /tt/home/timesten/
instances/instance1/diag/ttmesg.log --max-unchanged-stats=5; sleep
1; done
      exit 0
      EOF
    Requests:
      cpu:     100m
      memory:  20Mi
    Environment:
      TIMESTEN_HOME:        /tt/home/timesten/instances/instance1
      TT_OPERATOR_MANAGED:  1
      LD_LIBRARY_PATH:      /tt/home/timesten/instances/instance1/
ttclasses/lib:/tt/home/timesten/instances/instance1/install/lib:/tt/
home/timesten/instances/instance1/install/ttoracle_home/
instantclient_11_2:/tt/home/timesten/instances/instance1/install/
ttoracle_home/instantclient
    Mounts:
      /tt from tt-persistent (rw)
  Volumes:
   tt-agent:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  tte6d49f75-05ed-4b4b-9412-4a577ad19bbe
    Optional:    false
   tt-config:
    Type:               Projected (a volume that contains injected
data from multiple sources)
    ConfigMapName:      sample2
    ConfigMapOptional:  <nil>
Volume Claims:
  Name:          tt-persistent
  StorageClass:  oci
  Labels:        <none>
  Annotations:   <none>
  Capacity:      50G
  Access Modes:  [ReadWriteOnce]
Events:          <none>
```

You successfully modified the `sample2` TimesTenClassic object to use the new TimesTen container image. Let's continue the manual upgrade by upgrading the standby database.

Upgrade the standby database

Perform these steps to upgrade the standby database.

> **Note:**
>
> Even though you are upgrading the standby database, depending on your replication configuration, this may result in disruption on your active database. This may impact your applications. Perform the upgrade at the appropriate time.

1. Use the `kubectl get ttc` command to:
   - Determine which Pod is the standby. The active Pod is the Pod represented in the `ACTIVE` column. The standby Pod is the other Pod (not represented in the `ACTIVE` column). Therefore, for the `sample2` TimesTenClassic object, the active Pod is `sample2-0` and the standby Pod is `sample2-1`.
   - Ensure the state for the TimesTenClassic object (`sample2`, in this example) is `Normal`.

   ```
   kubectl get ttc sample2
   ```

   The output is the following.

   ```
   NAME      STATE    ACTIVE      AGE
   sample2   Normal   sample2-0   19h
   ```

2. To upgrade the standby to the new TimesTen image, delete the standby Pod (`sample2-1`, in this example).

   ```
   kubectl delete pod sample2-1
   ```

   The output is the following.

   ```
   pod "sample2-1" deleted
   ```

   Kubernetes automatically creates a new `sample2-1` Pod to replace the deleted Pod. The Operator configures the new `sample2-1` Pod as the standby Pod. This new Pod will now run the newly created TimesTen image.

3. Verify the standby is up and running and the state is `Normal`.

   ```
   kubectl get ttc sample2
   ```

Note that the state is initially `StandbyDown`.

```
NAME      STATE        ACTIVE     AGE
sample2   StandbyDown  sample2-0  19h
```

Wait a few minutes, then run the command again.

```
kubectl get ttc sample2
```

Note that the state has changed to `Normal`.

```
NAME      STATE   ACTIVE     AGE
sample2   Normal  sample2-0  19h
```

4. Verify that the standby is up and running again and that the active standby pair health is `Normal`. During the upgrade of the standby, your applications are not disrupted. Your applications can continue to use the active database.

```
kubectl describe ttc sample2
```

The output is similar to the following.

```
Name:         sample2
Namespace:    mynamespace
...
Kind:         TimesTenClassic
...
Spec:
  ...
  Ttspec:
    Additional Memory Request:  1Gi
    Automatic Memory Requests:  true
    Daemon Log CPU Request:     200m
    Daemon Log Memory Request:  20Mi
    Db Config Map:
      sample2
    Exporter CPU Request:     200m
    Exporter Memory Request:  200Mi
    Image:                    container-registry.oracle.com/
timesten/timesten:22.1.1.9.0
    Image Pull Policy:        Always
    Image Pull Secret:        sekret
    Image Upgrade Strategy:   Manual
    Memory Warning Percent:   90
    Storage Class Name:       oci
    Storage Size:             250G
Status:
  ...
  Classic Upgrade Status:
   ...
  High Level State:                 Normal
  ...
```

```
      Pod Status:
        Active:            true
        Admin User File:   true
        Cache Status:
          Cache Agent:         Not Running
          Cache UID Pwd Set:   true
          N Cache Groups:      0
        Cache User File:       false
        Cg File:               false
        Db Status:
          ...
        Name:                                              sample2-0
        Pod Status:
          Agent:                 Up
          Last Time Reachable:   1673213068
          Pod IP:                192.0.2.3
          Pod Phase:             Running
        Prev Active:             true
        Prev High Level State:   Healthy
        Prev Image:              container-registry.oracle.com/timesten/
    timesten:22.1.1.7.0
        Prev Intended State:
        Prev Ready:              true
        Ready:                   true
        Replication Status:
          Last Time Rep State Changed:  0
          Rep Agent:                    Running
          Rep Peer P State:             start
          Rep Scheme:                   Exists
          Rep State:                    ACTIVE
        Scaleout Status:
          Instance Type:  classic
        Schema File:      true
        Timesten Status:
          Daemon:         Up
          Instance:       Exists
          Release:        22.1.1.7.0
        Tt Pod Type:      Database
        Using Twosafe:    false
        Active:           false
        Admin User File:  true
        Cache Status:
          Cache Agent:         Not Running
          Cache UID Pwd Set:   true
          N Cache Groups:      0
        Cache User File:       false
        Cg File:               false
        Db Status:
          ...
        Name:                                                    sample2-1
        Pod Status:
          Agent:                 Up
          Last Time Reachable:   1673213068
          Pod IP:                192.0.2.4
          Pod Phase:             Running
        Prev Active:            false
        Prev High Level State:  Healthy
        Prev Image:             container-registry.oracle.com/timesten/
    timesten:22.1.1.7.0
        Prev Intended State:
        Prev Ready:             true
        Ready:                  true
        Replication Status:
          Last Time Rep State Changed:  1673212993
          Rep Agent:                    Running
          Rep Peer P State:             start
          Rep Scheme:                   Exists
          Rep State:                    STANDBY
        Scaleout Status:
          Instance Type:  classic
        Schema File:      true
        Timesten Status:
          Daemon:             Up
          Instance:           Exists
          Release:            22.1.1.9.0
        Tt Pod Type:          Database
        Using Twosafe:        false
      Prev High Level State:  StandbyCatchup
      Prev Reexamine:
      Prev Stop Managing:
      Rep Create Statement:   create active standby pair "sample2" on
    "sample2-0.sample2.mynamespace.svc.cluster.local", "sample2" on
    "sample2-1.sample2.mynamesoace.svc.cluster.local" NO RETURN store
    "sample2" on "sample2-0.sample2.mynamespace.svc.cluster.local" PORT
    4444 FAILTHRESHOLD 0 store "sample2" on
    "sample2-1.sample2.mynamespace.svc.cluster.local" PORT 4444
    FAILTHRESHOLD 0
      Rep Port:               4444
      Rep Start Fail Count:   0
      Standby Cache Agent:    Not Running
      Standby Down Standby AS:
        Async Id:           c3cf41af-61c2-438c-a4d0-f3dfbdb6043d
        Destroy Db:         true
        Id:                 8a9a22ee-deaa-4eb6-8e00-c0d8932af623
        Pod Name:           sample2-1
        Rep Duplicate:      true
        Start Rep Agent:    true
        Status:             complete
      Standby Perm In Use:  15291
      Standby Perm Size:    200
      Standby Rep Agent:    Running
      Status Version:       1.0
      Using Twosafe:        false
    Events:
      Type     Reason       Age    From      Message
      ----     ------       ----   ----      -------
      Normal   Upgrade      4m15s  timesten  Image updated, automatic
    upgrade disabled
      Normal   Info         3m24s  timesten  Pod sample2-1 Agent Down
      Normal   StateChange  3m24s  timesten  Pod sample2-1 is Not Ready
      Warning  StateChange  3m19s  timesten  TimesTenClassic was Normal, now
    ActiveTakeover
      Normal   StateChange  3m14s  timesten  TimesTenClassic was
    ActiveTakeover, now StandbyDown
      Normal   Info         2m19s  timesten  Pod sample2-1 Agent Up
      Normal   Info         2m19s  timesten  Pod sample2-1 Instance Exists
      Normal   Info         2m19s  timesten  Pod sample2-1 Daemon Down
      Normal   Info         2m19s  timesten  Pod sample2-1 Daemon Up
      Normal   Info         2m19s  timesten  Pod sample2-1 Database Unloaded
      Normal   Info         2m12s  timesten  Pod sample2-1 Database None
      Normal   Info         116s   timesten  Pod sample2-1 Database Loaded
      Normal   Info         116s   timesten  Pod sample2-1 RepAgent Not
    Running
      Normal   Info         116s   timesten  Pod sample2-1 RepScheme Exists
      Normal   StateChange  116s   timesten  Pod sample2-1 RepState IDLE
      Normal   Info         111s   timesten  Pod sample2-1 RepAgent Running
      Normal   StateChange  111s   timesten  TimesTenClassic was StandbyDown,
    now StandbyStarting
      Normal   StateChange  110s   timesten  TimesTenClassic was
    StandbyStarting, now StandbyCatchup
      Normal   StateChange  75s    timesten  Pod sample2-1 RepState STANDBY
      Normal   StateChange  75s    timesten  TimesTenClassic was
    StandbyCatchup, now Normal
      Normal   StateChange  70s    timesten  Pod sample2-1 is Ready
```

Note the following:

- The image is upgraded to the new release.
- The active database (`sample2-0`) is not upgraded to the new release.
- The standby database (`sample2-1`) is upgraded to the new release.

You have successfully upgraded the standby database. You are now ready to fail over from the active database to the standby.

Fail over

You must now fail over from the active database to the standby.

Note:

When you fail over, your active database will be taken down, and failed over **immediately**. Do not perform this procedure at the busiest time of your production day.

Before failing over, quiesce your applications on the active database. (You can also use the `ttAdmin -close` and the `ttAdmin -disconnect` commands. See Opening and closing the database for user connections and Disconnecting from a database in the *Oracle TimesTen In-Memory Database Operations Guide*.

To avoid potential data loss, use the `ttRepAdmin -wait` command to wait until replication is caught up, such that all transactions that were executed on the active database have been replicated to the standby database. See ttRepAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Once the standby is caught up, fail over from the active database to the standby by deleting the active Pod. When you delete the active Pod, the Operator automatically detects the failure and promotes the standby database to be the active. Client/server applications that are using the active database (`sample2-0`, in this example) are automatically reconnected to the new active database (`sample2-1`, in this example). Transactions in flight are rolled back. Prepared SQL statements will need to be re-prepared by the applications. See Handling failover and recovery for more information about client/server failover.

Kubernetes automatically creates a new `sample2-0` Pod to replace the deleted Pod. The Operator will configure the new Pod as the standby Pod. This new Pod will run the newly created TimesTen image.

Note:

You may want to perform this operation during a scheduled production outage.

1. Use the `kubectl delete` command to delete the active Pod (`sample2-0`, in this example).

   ```
   kubectl delete pod sample2-0
   ```

   The output is the following.

   ```
   pod "sample2-0" deleted
   ```

2. Use the `kubectl get` command to assess the state of the `sample2` TimesTenClassic object.

   ```
   kubectl get ttc sample2
   ```

   Note that the state is initially `ActiveDown`.

   ```
   NAME       STATE        ACTIVE    AGE
   sample2    ActiveDown   None      19h
   ```

   Wait a few minutes, then run the command again.

   ```
   kubectl get ttc sample2
   ```

   Note that the state has changed to `Normal`.

   ```
   NAME      STATE     ACTIVE      AGE
   sample2    Normal    sample2-1    19h
   ```

The Operator promoted `sample-1` to be the active.

3. Use the `kubectl describe` command to observe how the Operator recovers from the failure. The Operator promotes the standby database (`sample2-1`) to be active. Any applications that were connected to the `sample2-0` database are automatically reconnected to the `sample2-1` database by TimesTen. After a brief outage, the applications can continue to use the database. See Monitoring the health of the active standby pair of databases for information on the health and states of the active standby pair.

```
kubectl describe ttc sample2
```

The output is similar to the following.

```
Name:          sample2
Namespace:     mynamespace
...
Kind:          TimesTenClassic
...
Spec:
  ...
  Ttspec:
    Additional Memory Request:  1Gi
    Automatic Memory Requests:  true
    Daemon Log CPU Request:     200m
    Daemon Log Memory Request:  20Mi
    Db Config Map:
      sample2
    Exporter CPU Request:     200m
    Exporter Memory Request:  200Mi
    Image:                    container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Image Pull Policy:        Always
    Image Pull Secret:        sekret
    Image Upgrade Strategy:   Manual
    Memory Warning Percent:   90
    Storage Class Name:       oci
    Storage Size:             250G
Status:
  ...
  Classic Upgrade Status:
    ...
  High Level State:              Normal
  Last Event:                    0
  Last High Level State Switch:  1673214034
  Last Reconcile:                1673214897250
  Last Reconciling Operator:     timesten-operator-6f9d96bdfc-h22lm
  Observed Generation:           2
  Pod Status:
    Active:           false
    Admin User File:  true
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Cache User File:      false
    Cg File:              false
    Db Status:
      Db:  Loaded
      ...
    Name:                                                       sample2-0
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1673214897
      Pod IP:               192.0.2.3
      Pod Phase:            Running
    Prev Active:            false
    Prev High Level State:  Healthy
    Prev Image:             container-registry.oracle.com/timesten/
timesten:22.1.1.7.0
    Prev Intended State:    Active
    Prev Ready:             true
    Ready:                  true
    Replication Status:
      Last Time Rep State Changed:  1673213965
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Scaleout Status:
      Instance Type:  classic
    Schema File:      true
    Timesten Status:
      Daemon:         Up
      Instance:       Exists
      Release:        22.1.1.9.0
    Tt Pod Type:      Database
    Using Twosafe:    false
    Active:           true
    Admin User File:  true
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Cache User File:      false
    Cg File:              false
    Db Status:
      Db:  Loaded
      ...
    High Level State:                                           Healthy
    Initialized:                                                true
    Intended State:                                             Active
    Last High Level State Switch:                               1673214897
    Local Commit:                                               false
    Name:                                                       sample2-1
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1673214897
      Pod IP:               192.0.2.4
      Pod Phase:             Running
    Prev Active:             true
    Prev High Level State:  Healthy
    Prev Image:              container-registry.oracle.com/timesten/
timesten:22.1.1.7.0
    Prev Intended State:    Standby
    Prev Ready:              true
    Ready:                   true
    Replication Status:
      Last Time Rep State Changed:  1673212993
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    ACTIVE
    Scaleout Status:
      Instance Type:  classic
    Schema File:      true
    Timesten Status:
      Daemon:              Up
      Instance:            Exists
      Release:             22.1.1.9.0
    Tt Pod Type:           Database
    Using Twosafe:         false
  Prev High Level State:  StandbyDown
  Prev Reexamine:
  Prev Stop Managing:
  ...
Events:
  Type     Reason       Age   From      Message
  ----     ------       ----  ----      -------
  ...
  Warning  StateChange  15m   timesten  TimesTenClassic was Normal, now
ActiveDown
  Normal   Info         15m   timesten  Pod sample2-0 Agent Down
  Normal   StateChange  15m   timesten  Pod sample2-0 is Not Ready
  Normal   StateChange  15m   timesten  Pod sample2-0 is Not Active Ready
  Normal   Info         15m   timesten  Pod sample2-1 Database Updatable
  Normal   StateChange  15m   timesten  Pod sample2-1 RepState ACTIVE
  Normal   StateChange  15m   timesten  Pod sample2-1 is Not Ready
  Normal   StateChange  15m   timesten  TimesTenClassic was ActiveDown,
now ActiveTakeover
  Normal   StateChange  15m   timesten  Pod sample2-1 is Ready
  Normal   StateChange  15m   timesten  TimesTenClassic was
ActiveTakeover, now StandbyDown
  Normal   StateChange  15m   timesten  Pod sample2-1 is Active Ready
  Normal   Info         14m   timesten  Pod sample2-0 Agent Up
  Normal   Info         14m   timesten  Pod sample2-0 Instance Exists
  Normal   Info         14m   timesten  Pod sample2-0 Daemon Down
  Normal   Info         14m   timesten  Pod sample2-0 Daemon Up
  Normal   Info         14m   timesten  Pod sample2-0 Database Unloaded
  Normal   Info         14m   timesten  Pod sample2-0 Database None
  Normal   Info         14m   timesten  Pod sample2-0 Database Loaded
  Normal   Info         14m   timesten  Pod sample2-0 RepAgent Not Running
  Normal   Info         14m   timesten  Pod sample2-0 RepScheme Exists
  Normal   StateChange  14m   timesten  Pod sample2-0 RepState IDLE
  Normal   Info         14m   timesten  Pod sample2-0 RepAgent
Running
  Normal   StateChange  14m   timesten  Pod sample2-0 RepState
STANDBY
  Normal   StateChange  14m   timesten  Pod sample2-0 is Ready
  Normal   StateChange  14m   timesten  TimesTenClassic was
StandbyDown, now Normal
```

You successfully upgraded to a new release of TimesTen. The active and the standby Pods are running the new TimesTen image, which contains the new TimesTen release.

In this example, the `sample` and the `sample2` TimesTenClassic objects are now upgraded. The upgrade process is complete. Let's verify the active and the standby databases are running the new release of TimesTen.

Verify the active standby pair of databases are upgraded

After the upgrade process is complete for your TimesTenClassic objects, you can verify the active and standby databases are running the new release of TimesTen.

1. For the `sample` TimesTenClassic object, invoke a shell in the active Pod (`sample-1`, in this example). Then, run the TimesTen `ttVersion` utility to verify the release is the new release (`22.1.1.9.0`, in this example).

   ```
   kubectl exec -it sample-1 -c tt -- /bin/bash
   ```

   Run the TimesTen `ttVersion` utility.

   ```
   ttVersion
   ```

   The output is similar to the following.

   ```
   TimesTen Release 22.1.1.9.0 (64 bit Linux/x86_64) (instance1:6624)
   2022-09-29T07:40:22Z
     Instance admin: timesten
     Instance home directory: /tt/home/timesten/instances/instance1
     Group owner: timesten
     Daemon home directory: /tt/home/timesten/instances/instance1/info
     PL/SQL enabled.
   ```

   Exit from the shell.

   ```
   exit
   ```

2. Invoke a shell in the standby Pod (`sample-0`, in this example). Then, run the `ttVersion` utility to verify the release is the new release (`22.1.1.9.0`, in this example).

   ```
   kubectl exec -it sample-0 -c tt -- /bin/bash
   ```

Run the TimesTen `ttVersion` utility.

```
ttVersion
```

The output is similar to the following.

```
TimesTen Release 22.1.1.9.0 (64 bit Linux/x86_64) (instance1:6624)
2022-09-29T07:40:22Z
  Instance admin: timesten
  Instance home directory: /tt/home/timesten/instances/instance1
  Group owner: timesten
  Daemon home directory: /tt/home/timesten/instances/instance1/info
  PL/SQL enabled.
```

Exit from the shell.

```
exit
```

3. For the `sample2` TimesTenClassic object, invoke a shell in the active Pod (`sample2-1`, in this example). Then, use the `ttVersion` utility to verify the release is the new release (`22.1.1.9.0`, in this example).

```
 kubectl exec -it sample2-1 -c tt -- /bin/bash
```

Run the TimesTen `ttVersion` utility.

```
ttVersion
```

The output is similar to the following.

```
TimesTen Release 22.1.1.9.0 (64 bit Linux/x86_64) (instance1:6624)
2022-09-29T07:40:22Z
  Instance admin: timesten
  Instance home directory: /tt/home/timesten/instances/instance1
  Group owner: timesten
  Daemon home directory: /tt/home/timesten/instances/instance1/info
  PL/SQL enabled.
```

Exit from the shell.

```
exit
```

4. Invoke a shell in the standby Pod (`sample2-0`, in this example). Then, use the `ttVersion` utility to verify the release is the new release (`22.1.1.9.0`, in this example).

```
kubectl exec -it sample2-0 -c tt -- /bin/bash
```

Run the TimesTen `ttVersion` utility.

```
ttVersion
```

The output is similar to the following.

```
TimesTen Release 22.1.1.9.0 (64 bit Linux/x86_64) (instance1:6624)
2022-09-29T07:40:22Z
  Instance admin: timesten
  Instance home directory: /tt/home/timesten/instances/instance1
  Group owner: timesten
  Daemon home directory: /tt/home/timesten/instances/instance1/info
  PL/SQL enabled.
```

Exit from the shell.

```
exit
```

The upgrade to a new release of TimesTen is successful for the `sample` and the `sample2` TimesTenClassic objects. The active and the standby Pods for each TimesTenClassic object are running the new TimesTen image, which contains the new TimesTen release.

About upgrading direct mode applications

You cannot use this automated upgrade process to upgrade direct mode applications that are running in their own containers. The Operator does propagate the changes from a TimesTenClassic object to the associated StatefulSet, but the changes do not initiate the automated upgrade process. You must manually terminate the applications that are running in their containers. In so doing, StatefulSet spawns new containers to replace the original containers. These new containers run the newly specified TimesTen image. For more information about direct mode applications, see Using direct mode applications.

About failures during the upgrade process

If there are failures in any step of the upgrade process, a TimesTenClassic object enters the `ManualInterventionRequired` state. The remaining steps of the upgrade process are cancelled. You must manually fix the active/standby pair to return the pair to management by the Operator. Even when the pair is returned to automatic management, the remaining steps in the upgrade process are not automatically performed. See Understanding the ManualInterventionRequired state.

15
The TimesTen Operator Object Types

This chapter gives an overview of the TimesTenClassic and the TimesTenScaleout object types. The chapter describes the syntax for each object type.

Topics:

- Overview of the TimesTen Operator object types
- About the TimesTenClassic object type
- About the TimesTenScaleout object type

Overview of the TimesTen Operator object types

The TimesTen Operator installation defines the TimesTenClassic and the TimesTenScaleout object types to the Kubernetes cluster. An object of type TimesTenClassic describes the metadata for an active standby pair of TimesTen databases using TimesTen Classic. An object of type TimesTenScaleout describes the metadata for a TimesTen grid and a TimesTen database within the grid using TimesTen Scaleout. You can create as many of these TimesTenClassic and TimesTenScaleout objects as you like.

The definition of the TimesTenClassic and the TimesTenScaleout object types use the same basic format that the formal Kubernetes documentation uses to define objects that are built-in to Kubernetes. The facilities available in any given Kubernetes cluster depend on what release of Kubernetes the cluster is using. For information on the Kubernetes API documentation, see:

`https://kubernetes.io/docs/reference/kubernetes-api/`

The Kubernetes API reference documentation refers to a number of built-in Kubernetes types used in the definition of the TimesTenClassic and the TimesTenScaleout object types. A Kubernetes StatefulSet is of particular importance. The TimesTenClassic and the TimesTenScaleout object types are basically a wrapper around a StatefulSet type. For more information, see:

`https://kubernetes.io/docs/reference/kubernetes-api/`

Note:

All of the metadata is passed from the object to the StatefulSet.

About the TimesTenClassic object type

The definition of the TimesTenClassic object type uses the following object definitions:

- TimesTenClassic
- TimesTenClassicSpec

- TimesTenClassicSpecSpec
- TimesTenClassicSpecSpecPrometheus
- TimesTenClassicStatus

TimesTenClassic

An object of type TimesTenClassic describes the metadata for an active standby pair of TimesTen databases in TimesTen Classic.

The following table describes the syntax for the TimesTenClassic object type:

Table 15-1 TimesTenClassic syntax

Field	Type	Description
`apiVersion`	string	Versioned schema of this representation of an object. The value must be `timesten.oracle.com/v1`.
`kind`	string	Type of object (in this example, TimesTenClassic).
`metadata`	`ObjectMeta`	Metadata about the object, such as its name. For information on `ObjectMeta`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`spec`	`TimesTenClassicSpec`	Desired configuration of the TimesTen Pods and databases.
`status`	`TimesTenClassicStatus`	Current status of the Pods in this TimesTenClassic object as well as the status of various TimesTen components within those Pods. This data may be out of date by some window of time.

TimesTenClassicSpec

TimesTenClassicSpec appears in TimesTenClassic. The following table describes the syntax for TimesTenClassicSpec:

Table 15-2 TimesTenClassicSpec syntax

Field	Type	Description
`ttspec`	`TimesTenClassicSpecSpec`	TimesTen specific attributes.
`template`	`PodTemplateSpec`	Describes the Pods provisioned for the TimesTenClassic object. In addition to Pods specified by `template`, there are the `tt` and `daemonlog` containers that are automatically included in each Pod. TimesTen runs in the `tt` container. If you configure and use Prometheus, the `exporter` container is also included. For information on `PodTemplateSpec`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`

Table 15-2 (Cont.) TimesTenClassicSpec syntax

Field	Type	Description
`volumeClaimTemplates`	`PersistentVolumeClaim`	TimesTen automatically provisions PersistentVolumeClaims (PVCs) for `/tt` (and for `/ttlog`, if specified). If you have applications that are running in containers in the TimesTen Pods, and those applications require additional PVCs, specify them in this field. For information on `PersistentVolumeClaim`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`

TimesTenClassicSpecSpec

TimesTenClassicSpecSpec appears in TimesTenClassicSpec.

The following table describes the syntax for TimesTenClassicSpecSpec. There are some fields of type quantity. The specified value is of the same format as Kubernetes resource limits. For example, 200Gi, 200G, 1000Mi, 1000M, and so on.

Table 15-3 TimesTenClassicSpecSpec syntax

Field	Type	Description
`additionalMemoryRequest`	quantity	The amount of `memory` to request in addition to whatever is required for the TimesTen database. This memory is used for the TimesTen daemon, subdaemons, agents, and the Client/Server server. This value is added to `databaseMemorySize` that was either specified by you or calculated. The sum is the `memory` request to Kubernetes. The default is `1Gi`.
`agentGetTimeout`	integer	Time in seconds that the Operator waits for an https `GET` request to be processed by the TimesTen agent. This includes the TCP and the TLS times as well as the time it takes for the TimesTen agent to implement the `GET` request. The default is `60`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`agentPostTimeout`	integer	Time in seconds that the Operator waits for an https `POST` request to be processed by the TimesTen agent. This includes the TCP and the TLS times as well as the time it takes for the TimesTen agent to implement the `POST` request. The `POST` requests may take a long time and the time may be proportional to the size of the database. (An example is a `POST` request to duplicate a database from the active to the standby.) The default is `600`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the `POST` request to have failed.
`agentTCPTimeout`	integer	Time in seconds that the Operator waits for a TCP handshake when communicating with the TimesTen agent. The default is `10`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.
`agentTLSTimeout`	integer	Time in seconds that the Operator waits for a TLS (https) credential exchange when communicating with the TimesTen agent. The default is `10`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.
`automaticMemoryRequests`	boolean	Determines if the Operator attempts to set appropriate memory limits and requests for TimesTen Pods. Valid values: • `true` (default): The Operator attempts to set memory limits and requests. • `false`: The Operator does not set memory limits and requests.
`bothDownBehavior`	string	If the TimesTenClassic object enters the `BothDown` state, the Operator examines the `bothDownBehavior` setting to determine what to do. Acceptable values are `Best` (default) or `Manual`. See BothDown.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`cacheCleanup`	boolean	Determines if the metadata in the Oracle Database should be cleaned up when this TimesTenClassic object is deleted. Use for TimesTen Cache only. Valid values: • `true` (or not specified): The metadata is cleaned up. • `false`: The metadata is not cleaned up. See Dropping Oracle Database objects used by cache groups with autorefresh in the *Oracle TimesTen In-Memory Database Cache Guide*.
`daemonlogCPURequest`	quantity	The amount of `cpu` requested for the `daemonlog` container. The default is `200m` (one-fifth of a CPU).
`daemonLogMemoryRequest`	quantity	The amount of `memory` requested for the `daemonlog` container. The default is `20Mi`.
`daemonLogSidecar`	boolean	Determines if a daemon log container is created in each TimesTen Pod. This container writes the TimesTen daemon logs (from `ttmesg.log`) to `stdout`. This causes Kubernetes to record these logs. Valid values: • `true` (or not specified): A daemon log container is created. • `false`: A daemon log container is not created.
`databaseCPURequest`	quantity	Specify this value to tell the Operator how much CPU your `tt` containers require. This includes CPU used by the TimesTen daemon, subdaemons, replication agents, cache agents, and the Client/Server server. There is no default.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`databaseMemorySize`	quantity	You can specify this value to tell the Operator how much shared memory your database requires. If you specify a value, that value will be used. If you do not specify a value, the Operator attempts to determine the required size from the provided `db.ini` file. If the Operator cannot determine the database size, the value `580911104` is used. This is the size required for a default database with a `PermSize` of 200Mbyte, rounded up to 2Mi. This may be useful for experimentation, but is likely insufficient for production purposes. TimesTen recommends that you provide a `db.ini` file to the Operator by using a Configmap or Secret, and that you not specify `databaseMemorySize`. **Note:** If you provide a `db.ini` file by using an init container, you must specify `databaseMemorySize`.
`dbConfigMap`	array of strings	Name of one or more ConfigMaps to be included in a projected volume. This projected volume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbConfigMap` or `dbSecret`, you must place the metadata files into the `/ttconfig` directory by using other means. See Populating the /ttconfig directory.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`dbSecret`	array of strings	Name of one or more Secrets to be included in a projected volume. This projected volume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbSecret` or `dbConfigMap`, you must place the metadata files into the `/ttconfig` directory by using other means. See Populating the /ttconfig directory.
`exporterCPURequest`	quantity	The amount of `cpu` requested for the `exporter` container (if provisioned). The default is `200m` (one-fifth of a CPU).
`exporterMemoryRequest`	quantity	The amount of `memory` requested for the `exporter` container (if provisioned). The default is `200Mi`.
`image`	string	Name of the TimesTen image that is executed in the created containers. There is no default. You must specify the name of the `image`.
`imagePullPolicy`	string	Determines if and when Kubernetes pulls the TimesTen image from the image repository. Valid values: • `Always` • `IfNotPresent` (default) • `Never` Note: Values are case sensitive.
`imagePullSecret`	string	Image pull secret that is used to authenticate and give permission to Kubernetes to fetch the specified TimesTen image from its image repository. There is no default. You must specify the name of the image pull secret.
`imageUpgradeStrategy`	string	Determines if the Operator performs automated upgrades. Valid values: • `Auto` (or not specified): The Operator performs automated upgrades. • `Manual`: The Operator does not perform an automated upgrade. Values are case sensitive. See Performing Upgrades.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`logStorageClassName`	string	Name of the storage class that is used to request persistent volumes for the TimesTen database transaction log files. This field is optional.
`logStorageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store the TimesTen transaction logs, the primary determinant of what volumes to use is the `logStorageClassName` element that you specify. You can optionally specify a label selector by using the `logStorageSelector` element. This label selector further filters the set of volumes. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`logStorageSize`	string	Amount of storage to be provisioned for the TimesTen transaction logs. For information on determining the amount of storage needed for the transaction log files, see Storage provisioning for TimesTen in the *Oracle TimesTen In-Memory Database Operations Guide* The default is `50Gi`. This default value may be suitable when you are experimenting with the product or using it for demonstration purposes. However, in a production environment, consider choosing a value greater than `50Gi`. The examples in this book assume a production environment and use a value of `250Gi`.
`memoryWarningPercent`	integer	At runtime, if a container's memory usage is more than its percentage of its limit (both as reported by cgroups), the Operator generates Events to inform you of this occurrence. The memory usage refers to the container's `memory` allocation. The default is `90`.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`pollingInterval`	integer	Determines how often (expressed in seconds) that the Operator checks the status of the TimesTenClassic active standby pair object. For example, if you set this value to `10`, the Operator checks the status of the TimesTenClassic object every ten seconds. This value interacts with `unreachableTimeout`. The `pollingInterval` value should be smaller than the `unreachableTimeout` value. The value must be a positive integer (greater than `0`). The default is `5`.
`prometheus`	`TimesTenClassicSpecSpecPrometheus`	Determines if the TimesTen Exporter is deployed. If specified, the Exporter is deployed. The attributes for the `prometheus` object are defined in TimesTenClassicSpecSpecPrometheus.
`reexamine`	string	When a TimesTenClassic object is in the `ManualInterventionRequired` state, the Operator examines the `reexamine` value every `pollingInterval` seconds. If the value has changed since the last iteration for this object, the Operator examines the state of the TimesTen containers for this object. See Understanding the ManualInterventionRequired state and Bringing up one database.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description

`repCreateStatement`	string	The `repReturnServiceAttribute` and the `repStoreAttribute` datum provide some control over the `CREATE ACTIVE STANDBY` statement that you use to configure your active standby pair replication scheme. However, these elements do not provide a mechanism to set all the replication options.
		The `repCreateStatement` datum provides more control over the active standby pair replication configuration. If you choose to define a replication scheme, you must choose either the `repCreateStatement` approach or the `repReturnServiceAttribute` and the `repStoreAttribute` approach. You cannot use both approaches simultaneously in a single TimesTenClassic object definition. For example, you cannot use the `repCreateStatement` element and the `repReturnServiceAttribute` element in a single TimesTenClassic object definition. However, you can use the `repReturnServiceAttribute` and the `repStoreAttribute` elements in a single TimesTenClassic object definition.
		Example of using the `repCreateStatement` element:
		`apiVersion: timesten.oracle.com/v1`
		`kind: TimesTenClassic`
		`metadata:`
		` name: sample`
		`spec:`
		`  ttspec:`
		` repCreateStatement: |`
		`      create active standby pair`
		`        "{{tt-name}}" on "{{tt-`
		`node-0}}",`
		`        "{{tt-name}}" on "{{tt-`
		`node-1}}"`
		`      RETURN TWOSAFE`
		`      store "{{tt-name}}" on`
		`"{{tt-node-0}}"`
		`        PORT {{tt-rep-port}}`
		`FAILTHRESHOLD 10 TIMEOUT 5`
		`        DISABLE RETURN ALL 10`
		`      store "{{tt-name}}" on`
		`"{{tt-node-1}}"`
		`        PORT {{tt-rep-port}}`
		`FAILTHRESHOLD 10 TIMEOUT 5`
		`        DISABLE RETURN ALL 10`

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
		The Operator does the substitutions for you. • `{{tt-name}}`: The name of the TimesTenClassic object. (For example, `sample`.) • `{{tt-node-0}}`: The fully qualified DNS name of the `-0` Pod for the TimesTenClassic object. (For example, `sample-0.sample.mynamespace.svc.cluster.local`.) • `{{tt-node-1}}`: The fully qualified DNS name of the `-1` Pod for the TimesTenClassic object. (For example, `sample-1.sample.mynamespace.svc.cluster.local`.) • `{{tt-rep-port}}`: The TCP port either chosen by the Operator or specified in the `repPort` datum. When you use the `repCreateStatement` element, you have nearly complete control over the replication configuration. The Operator executes the statement you define (after substituting a number of values into it). Since the Operator is using the `CREATE` statement that you define, ensure that the statement you specify is correct and appropriate. If the creation of your active standby pair replication scheme fails, your TimesTenClassic object transitions from the `Initializing` state to the `Failed` state. You must then delete the TimesTenClassic object to clean up the resources it holds. See Monitoring the health of the active standby pair of databases. The configuration has the following restrictions: • Must be an active standby pair. • Must not be configured with subscribers. See CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* and Defining an active standby pair replication scheme in the *Oracle TimesTen In-Memory Database Replication Guide*.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`replicationCipherSuite`	string	Determines the encryption algorithm to be used by TimesTen replication. If specified, replication traffic is encrypted. You can specify one or more values. Possible values include the following: • `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` • `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` See About using certificates with TimesTen in the *Oracle TimesTen In-Memory Database Security Guide* for more information.
`replicationSSLMandatory`	integer	Determines if SSL encryption is mandatory for replication. Valid values: • `0` (or not specified): SSL encryption is not mandatory for replication. • `1`: SSL encryption is mandatory for replication. This value is only examined if `replicationCipherSuite` is specified. See About using certificates with TimesTen in the *Oracle TimesTen In-Memory Database Security Guide* for more information.
`repPort`	integer	TCP port used for replication. The default is `4444`.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`repReturnServiceAttribute`	string	You can use the `repReturnServiceAttribute` element to specify the *`ReturnServiceAttribute`* clause. This clause is part of the syntax for the `CREATE ACTIVE STANDBY PAIR` statement. The information you specify is included in your active standby pair's `CREATE ACTIVE STANDBY PAIR` statement by the Operator. Do not specify the `repReturnServiceAttribute` element if you have specified the `repCreateStatement` element. If you do not specify the `repReturnServiceAttribute` element (or the `repCreateStatement` element), the default is `NO RETURN`. See CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* and Defining an active standby pair replication scheme in the *Oracle TimesTen In-Memory Database Replication Guide* for information on the `CREATE ACTIVE STANDBY PAIR` statement and the *`ReturnServiceAttribute`* clause.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`repStoreAttribute`	string	You can use the `repStoreAttribute` element to specify the *`StoreAttribute`* clause. This clause is part of the `CREATE ACTIVE STANDBY PAIR` statement. The information you specify is included in your active standby pair's `CREATE ACTIVE STANDBY PAIR` statement by the Operator. Do not specify the `repStoreAttribute` element if you have specified the `repCreateStatement` element. If you do not specify the `repStoreAttribute` element (or the `repCreateStatement` element), the default is: `PORT` *`repPort`* `FAILTHRESHOLD 0`. If you specify the `repStoreAttribute`, you must specify the port. This port is used by replication. The port must match the port provided in the `repPort` element (or must match the default value if `repPort` is not specified). If the ports do not match, the TimesTenClassic object enters the `Failed` state. See CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* and Defining an active standby pair replication scheme in the *Oracle TimesTen In-Memory Database Replication Guide* for information on the `CREATE ACTIVE STANDBY PAIR` statement and the *`StoreAttribute`* clause.
`stopManaging`	string	If you change the value of `stopManaging` for the TimesTenClassic object, the Operator places the object in the `ManualInterventionRequired` state. See Understanding the ManualInterventionRequired state and Bringing up one database.
`storageClassName`	string	Name of the storage class that is used to request persistent volumes for the TimesTen database. There is no default. You must specify the name of the storage class.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`storageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store a TimesTen database, the primary determinant of what volumes to use is the `StorageClassName` that you specify. You can optionally specify a label selector by using the `storageSelector` field. This label selector further filters the set of volumes. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`storageSize`	string	Amount of storage to be provisioned for TimesTen and the database. For information on determining the amount of storage needed for TimesTen, see Storage provisioning for TimesTen in the *Oracle TimesTen In-Memory Database Operations Guide* . The default is `50Gi`. This default value may be suitable when you are experimenting with the product or using it for demonstration purposes. However, in a production environment, consider choosing a value greater than `50Gi`. The examples in this book assume a production environment and use a value of `250Gi`.
`unreachableTimeout`	integer	Number of seconds that a TimesTen instance or TimesTen database is unavailable before the Operator takes action to fail over or otherwise recover from the issue. This value interacts with `pollingInterval`. The `pollingInterval` value should be smaller than the `unreachableTimeout` value. The value must be a positive integer (greater than `0`). The default is `30`.

Table 15-3 (Cont.) TimesTenClassicSpecSpec syntax

Field	Type	Description
`upgradeDownPodTimeout`	integer	Maximum amount of seconds that the TimesTenClassic object remains in the `WaitingForActive` state. After this period of time, if the TimesTenClassic object is still in the `WaitingForActive` state, it transitions to the `ManualInterventionRequired` state. The default is `0` (which means there is no timeout. The TimesTenClassic object waits forever, if required). For information on the `WaitingForActive` and the `ManualInterventionRequired` states, see Monitoring the health of the active standby pair of databases.

TimesTenClassicSpecSpecPrometheus

TimesTenClassicSpecSpecPrometheus appears in TimesTenClassicSpecSpec. The following table describes the syntax for TimesTenClassicSpecSpecPrometheus.

Table 15-4 TimesTenClassicSpecSpecPrometheus syntax

Field	Type	Description
`port`	integer	Port on which the TimesTen Exporter listens. The default is `8888`.
`limitRate`	integer	Limit of GET requests per minute that the Exporter accepts. The value can be any integer value from `1` to `15`. The default is `10`.
`insecure`	boolean	Determines if the Exporter is started with no authentication or with client certificate authentication. Valid values: • `true`: The Exporter is started with no authentication and serves data by HTTP.. • `false` (or not specified): The Exporter is started with client certificate authentication and serves data by HTTPS. For information about the Exporter and its authentication, see About the TimesTen exporter in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

Table 15-4 (Cont.) TimesTenClassicSpecSpecPrometheus syntax

Field	Type	Description
`certSecret`	string	Name of the Kubernetes Secret that contains the Oracle Wallet used by the Exporter for client certificate authentication. If you are using client certificate authentication, you must specify this field and supply a Kubernetes Secret. Otherwise, do not specify this field.

TimesTenClassicStatus

TimesTenClassicStatus appears in TimesTenClassic. The Operator stores various persistent information in TimesTenClassicStatus.

The output of the `kubectl get` and `kubectl describe` commands display information in TimesTenClassicStatus. This information includes the following:

- `awtBehindMb`: a field that is present only if AWT (Asynchronous WriteThrough) is in use. The field represents how many megabytes of log is present in TimesTen that has not yet been pushed to Oracle Database. For more information on AWT cache group, see Overview of cache groups in the *Oracle TimesTen In-Memory Database Cache Guide*.
- High Level state of the Active Standby Pair: a string that describes the High Level state of the active standby pair.
- Detailed state of TimesTen in each Pod, which includes the following:
  - Is the TimesTen agent running?
  - Is the TimesTen main daemon running?
  - Is the TimesTen replication agent running?
  - Is the TimesTen cache agent running?
  - Is there a database in the instance?
  - Is the database loaded?
  - Is the database updatable or read only?
  - Is there a replication scheme in the database?
  - What is the replication state of this database?
  - What does this database think the replication state of its peer is?
  - What is the role for TimesTen in this Pod (active or standby)?
  - What is the High Level state of the Pod?

Note:

Unknown values can occur if, for example, the agent is not running or a Pod is unavailable.

About the TimesTenScaleout object type

The TimesTenScaleout object type is defined using the following object definitions:

- TimesTen Scaleout
- TimesTenScaleoutSpec
- TimesTenScaleoutSpecSpec
- TimesTenScaleoutSpecSpecPrometheus
- TimesTenScaleoutStatus

TimesTen Scaleout

An object of type TimesTenScaleout describes the metadata for a TimesTen grid and the TimesTen database within the grid in TimesTen Scaleout.

The following table describes the syntax for the TimesTenScaleout object type:

Table 15-5 TimesTenScaleout syntax

Field	Type	Description
`apiVersion`	string	Versioned schema of this representation of an object. The value must be `timesten.oracle.com/v1`.
`kind`	string	Type of object (in this example, TimesTenScaleout).
`metadata`	`ObjectMeta`	Metadata about the object, such as its name. For information on `ObjectMeta`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`spec`	`TimesTenScaleoutSpec`	Desired configuration of the TimesTen Scaleout grid and the databases within the grid.
`status`	`TimesTenScaleoutStatus`	Current status of the Pods in this TimesTenScaleout object as well as the status of various TimesTen components within those Pods. This data may be out of date by some window of time.

TimesTenScaleoutSpec

TimesTenScaleoutSpec appears in TimesTenScaleout. The following table describes the syntax for TimesTenScaleoutSpec:

Table 15-6 TimesTenScaleoutSpec syntax

Field	Type	Description
`ttspec`	`TimesTenScaleoutSpecSpec`	Specific TimesTen attributes for deploying a grid in TimesTen Scaleout.

Table 15-6 (Cont.) TimesTenScaleoutSpec syntax

Field	Type	Description
`dataTemplate`	Array of `PodTemplateSpec`	Array of specifications for the Pods that contain the TimesTen Scaleout data instances. This field is optional. If specified, there must be `k` entries in the array. Each entry is the template for the Pods in a different data space. For example, if `k` is set to `2`, then the first entry is for data space 1, and the second entry is for data space 2. Use `dataTemplate` to pass affinity and other settings to Kubernetes. For information about `PodTemplateSpec`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`mgmtTemplate`	`PodTemplateSpec`	Specification for the Pod that contains the TimesTen Scaleout management instance. Use `mgmtTemplate` to pass affinity and other settings to Kubernetes. For information about `PodTemplateSpec`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`zookeeperTemplate`	Array of `PodTemplateSpec`	Specification for the Pods that contains the TimesTen Scaleout ZooKeeper instances. Use `zookeeperTemplate` to pass affinity and other settings to Kubernetes. For information about `PodTemplateSpec`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`volumeClaimTemplates`	`PersistentVolumeClaim`	TimesTen automatically provisions PersistentVolumeClaims (PVCs) for `/tt` (and for /`ttlog`, if specified). If you have applications that are running in containers in the TimesTen Pods, and those applications require additional PVCs, specify them in this field. For information on `PersistentVolumeClaim`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`

TimesTenScaleoutSpecSpec

TimesTenScaleoutSpecSpec appears in TimesTenScaleoutSpec.

The following table describes the syntax for TimesTenScaleoutSpecSpec. There are some fields of type quantity. The specified value is of the same format as Kubernetes resource limits. For example, 200Gi, 200G, 1000Mi, 1000M, and so on.

Table 15-7 TimesTenScaleoutSpecSpec

Field	Type	Description
`additionalMemoryRequest`	quantity	The amount of `memory` to request in addition to whatever is required for an element of the TimesTen database. This memory is used for the TimesTen daemon, subdaemons, agents, and the Client/Server server. This value is added to `databaseMemorySize` that was either specified by you or calculated. The sum is the `memory` request to Kubernetes. The default is `1Gi`.
`agentGetTimeout`	integer	Time in seconds that the Operator waits for an https `GET` request to be processed by the TimesTen agent. This includes the TCP and the TLS times as well as the time it takes for the TimesTen agent to implement the `GET` request. The default is `60`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.
`agentPostTimeout`	integer	Time in seconds that the Operator waits for an https `POST` request to be processed by the TimesTen agent. This includes the TCP and the TLS times as well as the time it takes for the TimesTen agent to implement the `POST` request. The `POST` requests may take a long time and the time may be proportional to the size of the database. (An example is a `POST` request to duplicate a database from the active to the standby.) The default is `600`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the `POST` request to have failed.
`agentTCPTimeout`	integer	Time in seconds that the Operator waits for a TCP handshake when communicating with the TimesTen agent. The default is `10`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`agentTLSTimeout`	integer	Time in seconds that the Operator waits for a TLS (https) credential exchange when communicating with the TimesTen agent. The default is `10`. A value of `0` indicates that there is no timeout. If the timeout is exceeded, the Operator considers the agent to be down.
`automaticMemoryRequests`	boolean	Determines if the Operator attempts to set appropriate memory limits and requests for TimesTen Pods. Valid values: • `true` (default): The Operator attempts to set memory limits and requests. • `false`: The Operator does not set memory limits and requests.
`cacheCleanup`	boolean	Determines if the metadata in the Oracle Database should be cleaned up when this TimesTenClassic object is deleted. Use for TimesTen Cache only. Valid values: • `true` (or not specified): The metadata is cleaned up. • `false`: The metadata is not cleaned up. See Dropping Oracle Database objects used by cache groups with autorefresh in the *Oracle TimesTen In-Memory Database Cache Guide*.
`daemonlogCPURequest`	quantity	The amount of `cpu` requested for the `daemonlog` container. The default is `200m` (one-fifth of a CPU).
`daemonLogMemoryRequest`	quantity	The amount of `memory` requested for the `daemonlog` container. The default is `20Mi`.
`daemonLogSidecar`	boolean	Determines if a daemon log container is created in each TimesTen Pod. This container writes the TimesTen daemon logs (from `ttmesg.log`) to `stdout`. This causes Kubernetes to record these logs. Valid values: • `true` (or not specified): A daemon log container is created. • `false`: A daemon log container is not created.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`databaseCPURequest`	Quantity	Specify this value to tell the Operator how much CPU your `tt` containers require. This includes CPU used by the TimesTen daemon, subdaemons, replication agents, cache agents, and the Client/Server server. There is no default.
`databaseMemorySize`	Quantity	You can specify this value to tell the Operator how much shared memory an element of your database requires. If you specify a value, that value will be used. If you do not specify a value, the Operator attempts to determine the required size from the provided `db.ini` file. If the Operator cannot determine the database size, the value `580911104` is used. This is the size required for a default database with a `PermSize` of 200Mbyte, rounded up to 2Mi. This may be useful for experimentation, but is likely insufficient for production purposes. TimesTen recommends that you provide a `db.ini` file to the Operator by using a Configmap or Secret, and that you not specify `databaseMemorySize`. **Note:** If you provide a `db.ini` file by using an init container, you must specify `databaseMemorySize`.
`dataStorageClassName`	string	Name of the storage class that is used to request persistent volumes for the elements of the TimesTen database in the grid. If not specified, the default is the value of `storageClassName`.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`dataStorageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store the elements of a TimesTen database in the grid, the primary determinant of what volumes to use is the `dataStorageClassName` that you specify. You can optionally specify a label selector by using the `dataStorageSelector` field. This label selector further filters the set of volumes. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`dataStorageSize`	string	Amount of storage to be provisioned for each element of the TimesTen database in the grid. The default is `50Gi`. This default value may be suitable when you are experimenting with the product or using it for demonstration purposes. However, in a production environment, consider choosing a value greater than `50Gi`. The examples in this book assume a production environment and use a value of `250Gi`.
`dbConfigMap`	array of strings	Name of one or more ConfigMaps to be included in a projected volume. This projected volume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbConfigMap` or `dbSecret`, you must place the metadata files into the `/ttconfig` directory by using other means. See Populating the /ttconfig directory.
`dbSecret`	array of strings	Name of one or more Secrets to be included in a projected volume. This projected volume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbSecret` or `dbConfigMap`, you must place the metadata files into the `/ttconfig` directory by using other means. See Populating the /ttconfig directory.
`exporterCPURequest`	quantity	The amount of `cpu` requested for the `exporter` container (if provisioned). The default is `200m` (one-fifth of a CPU).
`exporterMemoryRequest`	quantity	The amount of `memory` requested for the `exporter` container (if provisioned). The default is `200Mi`.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`image`	string	Name of the TimesTen image that is executed in the created containers. There is no default. You must specify the name of the `image`.
`imagePullPolicy`	string	Determines if and when Kubernetes pulls the TimesTen image from the image repository. Valid values: • `Always` • `IfNotPresent` (default) • `Never` Note: Values are case sensitive.
`imagePullSecret`	string	Image pull secret that is used to authenticate and give permission to Kubernetes to fetch the specified TimesTen image from its image repository. There is no default. You must specify the name of the image pull secret.
`k`	integer	K-safety value for this TimesTen grid. This value determines the number of copies of data for your TimesTen database. This value also determines the number of StatefulSets that the TimesTen Operator creates. A StatefulSet provides the Pods that are used to implement a single data space in the grid. For example, if you set `k` to `2`, the Operator creates two StatefulSets. One StatefulSet provides the Pods for the data instances in data space one. The second StatefulSet provides the Pods for the data instances in data space two. The Operator also creates a StatefulSet for the management instance and a StatefulSet for ZooKeeper. The default is `2`. For more information on K-safety, see K-safety in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
`logStorageClassName`	string	Name of the storage class that is used to request persistent volumes for the TimesTen database transaction log files. This field is optional and is valid only for data instances.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`logStorageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store the TimesTen transaction log files, the primary determinant of what volumes to use is the `logStorageClassName` that you specify. You can optionally specify a label selector by using the `logStorageSelector` field. This label selector further filters the set of volumes. Valid only for data instances. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`logStorageSize`	string	Amount of storage to be provisioned for the TimesTen transaction log files. This value is for each element of the TimesTen database in the grid. Valid only for data instances.
`memoryWarningPercent`	integer	At runtime, if a container's memory usage is more than its percentage of its limit (both as reported by cgroups), the Operator generates Events to inform you of this occurrence. The memory usage refers to the container's `memory` allocation. The default is `90`.
`mgmtCPURequest`	quantity	The amount of `cpu` requested for the `tt` container of management instances. The default is `1`.
`mgmtMemoryRequest`	quantity	The amount of `memory` requested for the `tt` container of management instances. The default is `1Gi`.
`mgmtStorageClassName`	string	Name of the storage class that is used to request persistent volumes for the database of the management instance. If not specified, the default is the value of `storageClassName`.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`mgmtStorageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store the database of the management instance, the primary determinant of what volumes to use is the `mgmtStorageClassName` that you specify. You can optionally specify a label selector by using the `mgmtstorageSelector` field. This label selector further filters the set of volumes. If you do not specify `mgmtstorageSelector`, the value is the value of `StorageSelector`. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`mgmtStorageSize`	string	Amount of storage to be provisioned for the database of the management instance. The default is `50Gi`.
`nReplicaSets`	integer	Number of replica sets in the grid. A replica set contains `k` elements (where each element is an exact copy of the other elements in the replica set). The `nReplicaSets` value also determines the number of replicas for each StatefulSet. For example, if you set `k` to `2`, the TimesTen Operator creates two StatefulSets for the data instances. If you set `nReplicaSets` to `3`, each StatefulSet contains three replicas, and the total number of replica sets in the grid is three. The default is `1`. For more information on replica sets, see Understanding replica sets in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
`nZookeeper`	integer	Number of ZooKeeper Pods to provision in a StatefulSet. Valid values: • `1` • `3` (default) • `5`

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`pollingInterval`	integer	Determines how often (expressed in seconds) that the Operator checks the status of the TimesTenScaleout object. For example, if you set this value to `10`, the Operator checks the status of the TimesTenScaleout object every ten seconds. The value must be a positive integer (greater than `0`). The default is `5`.
`prometheus`	`TimesTenScaleoutSpecSpecPrometheus`	Determines if the TimesTen Exporter is deployed. If specified, the Exporter is deployed. The attributes for the `prometheus` object are defined in TimesTenScaleoutSpecSpecPrometheus.
`reexamine`	string	When a TimesTenScaleout object is in the `ManualInterventionRequired` state, the Operator examines the `reexamine` value every `pollingInterval` seconds. If the value has changed since the last iteration for this object, the Operator reexamines the TimesTenScaleout object and attempts to resume management of the object.
`replicaSetRecovery`	string	Controls the behavior of the Operator when a total replica set failure occurs. Valid values: • `Restart` (default): The Operator forcibly unloads and reloads the database when a total replica set failure occurs. • `Manual`: The Operator changes the state of the TimesTenScaleout object to `ManualInterventionRequired` when a total replica set failure occurs.
`stopManaging`	string	If you change the value of `stopManaging` for the TimesTenScaleout object, the Operator places the object in the `ManualInterventionRequired` state.

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`storageClassName`	string	Name of the storage class that is used to request persistent volumes for the TimesTen database. If the value for `storageClassName` is the same as the value for `dataStorageClassName`, for `mgmtStorageClassName`, and for `zookeeperStorageClassName`, you can just specify `storageClassName`. In this case, the value for `dataStorageClassName`, for `mgmtStorageClassName`, and for `zookeeperStorageClassName` is set to the value of `storageClassName`.
`storageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store an element of a TimesTen database in a grid, the primary determinant of what volumes to use is the `StorageClassName` that you specify. You can optionally specify a label selector by using the `storageSelector` field. This label selector further filters the set of volumes. If the value for `storageSelector` is the same as the value for `dataStorageSelector`, for `mgmtStorageSelector`, and for `zookeeperStorageSelector`, you can just specify `storageSelector`. In this case, the value for `dataStorageSelector`, for `mgmtStorageSelector`, and for `zookeeperStorageSelector` is set to the value of `storageSelector`. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`

Table 15-7 (Cont.) TimesTenScaleoutSpecSpec

Field	Type	Description
`storageSize`	string	Amount of storage to be used for each element of a TimesTen database in the grid. If the value for `storageSize` is the same as the value for `dataStorageSize`, for `mgmtStorageSize`, and for `zookeeperStorageSize`, you can just specify `storageSize`. In this case, the value for `dataStorageSize`, for `mgmtStorageSize`, and for `zookeeperStorageSize` is set to the value of `storageSize`. The default is `50Gi`. For information on determining the amount of storage needed for TimesTen, see Storage provisioning for TimesTen in the *Oracle TimesTen In-Memory Database Operations Guide* .
`zookeeperCPURequest`	quantity	The amount of `cpu` requested for the `zookeeper` container of Zookeeper Pods. The default is `500m`.
`zookeeperMemoryRequest`	quantity	The amount of `memory` requested for the `zookeeper` container of Zookeeper Pods. The default is `1Gi`.
`zookeeperStorageClassName`	string	Name of the storage class that is used to request persistent volumes for ZooKeeper's persistent data. If not specified, the default is the value of `storageClassName`.
`zookeeperStorageSelector`	`metav1.LabelSelector`	When choosing to use a persistent volume to store ZooKeeper's persistent data, the primary determinant of what volumes to use is the `zookeeperStorageClassName` that you specify. You can optionally specify a label selector by using the `zookeeperStorageSelector` field. This label selector further filters the set of volumes. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`zookeeperStorageSize`	string	Amount of storage to be provisioned for ZooKeeper's persistent data . The default is `50Gi`.

TimesTenScaleoutSpecSpecPrometheus

TimesTenScaleoutSpecSpecPrometheus appears in TimesTenScaleoutSpecSpec . The following table describes the syntax for TimesTenScaleoutSpecSpecPrometheus.

Table 15-8 TimesTenScaleoutSpecSpecPrometheus syntax

Field	Type	Description
`port`	integer	Port on which the TimesTen Exporter listens. The default is `8888`.
`limitRate`	integer	Limit of GET requests per minute that the Exporter accepts. The value can be any integer value from `1` to `15`. The default is `10`.
`insecure`	boolean	Determines if the Exporter is started with no authentication or with client certificate authentication. Valid values: • `true`: The Exporter is started with no authentication and serves data by HTTP. • `false` (or not specified): The Exporter is started with client certificate authentication and serves data by HTTPS. For information about the Exporter and its authentication, see About the TimesTen exporter in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.
`certSecret`	string	Name of the Kubernetes Secret that contains the Oracle Wallet used by the Exporter for client certificate authentication. If you are using client certificate authentication, you must specify this field and supply a Kubernetes Secret. Otherwise, do not specify this field.

TimesTenScaleoutStatus

TimesTenScaleoutStatus appears in TimesTen Scaleout. The Operator stores various persistent information in TimesTenScaleoutStatus.

The output of the `kubectl get` and `kubectl describe` commands display information in TimesTenScaleoutStatus.

The information includes the following:

- High Level state of the TimesTen grid and the database within the grid.
- Detailed state of TimesTen in each Pod, including:

- Are the TimesTen agents running?
- Are the TimesTen daemons running?
- Is there a database in the instance?
- Is the database loaded?
- Is the database updatable or read only?
- What is the High Level state of the Pods?

Note:

Unknown values can occur if, for example, the agent is not running or a Pod is unavailable.

16 Dockerfile ARGs

Images containing TimesTen and its prerequisites are available at `container-registry.oracle.com`. You can use these images as is with the TimesTen Kubernetes Operator. You also have the choice to build your own TimesTen container image by using the Dockerfile provided in the `/image` directory of the `operator.zip` distribution. The `operator.zip` distribution is located in the `/kubernetes` directory of the TimesTen distribution. See Unpack the TimesTen and the TimesTen Kubernetes Operator distributions.

The Dockerfile supports a number of ARGs. These ARGs let you override the attributes of the Dockerfile (and its resultant image). You supply these ARGs on the `docker build` command line.

Table 16-1 describes the supported ARGs:

Table 16-1 Dockerfile ARGs

ARG name	Default value	Description
`TT_BASE`	`container-registry.oracle.com/os/oraclelinux:8`	Name of the base image.
`UNZIP_BASE`	`container-registry.oracle.com/os/oraclelinux:8`	Name of the image that is used to unzip the TimesTen distribution.
`TT_DISTRO`	`timesten221190.server.linux8664.zip`	Name of the TimesTen distribution that you are including in the container image. If you are building the TimesTen container image, you must supply this ARG on the `docker build` command line.
`TT_RELEASE`	`22.1.1.9.0`	Release number (in dotted decimal format) of the TimesTen release that is included in `$TT_DISTRO`. If you are building the TimesTen container image, you must supply this ARG on the `docker build` command line.
`TT_USER`	`timesten`	Name of the Linux user that is created in the container image. This is the user who runs TimesTen. To define a different user to run TimesTen, supply this ARG on the `docker build` command line.

Table 16-1 (Cont.) Dockerfile ARGs

ARG name	Default value	Description
`TT_UID`	`3429`	The numeric UID of `$TT_USER`. To define a different UID for `$TT_USER`, supply this ARG on the `docker build` command line.
`TT_GROUP`	`timesten`	Name of the Linux group that is created in the container image. This is the name of the TimesTen users group. To define a different name for the TimesTen group, supply this ARG on the `docker build` command line.
`TT_GID`	`3429`	The numeric GID of `$TT_GROUP`. To define a different GID for `$TT_GROUP`, supply this ARG on the `docker build` command line.
`TT_EXTRA_LINUX_PACKAGES`	There is no default.	Additional Linux packages that you want installed in the container image by the `yum install` command.

A

Active Standby Pair Example

This appendix provides an example showing you the complete process for deploying and running your active standby pair of TimesTen databases in the Kubernetes cluster. After the databases are up and running, the example demonstrates how the Operator controls and manages the databases. If the active database fails, the Operator performs the necessary tasks to failover to the standby database, making that standby database the active one. The example concludes with procedures to delete the TimesTen databases and to stop the Operator.

- Before you begin
- Create the ConfigMap object
- Create the TimesTenClassic object
- Monitor deployment
- Verify the existence of the underlying objects
- Verify the connection to the active TimesTen database
- Recover from failure
- Cleanup

Before you begin

Review the following sections and complete if you have not already done so.

1. Prerequisites
2. Setting Up the Environment

Create the ConfigMap object

This section creates the `sample` ConfigMap. This ConfigMap contains the `db.ini`, the `adminUser`, and the `schema.sql` metadata files. This ConfigMap will be referenced when you define the TimesTenClassic object. See "Overview of the configuration metadata and the Kubernetes facilities" for information on the configuration files and the ConfigMap facility.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_sample` subdirectory. (The `cm_sample` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p cm_sample
   ```

2. Navigate to the ConfigMap directory.

   ```
   % cd cm_sample
   ```

3. Create the `db.ini` file in this ConfigMap directory (`cm_sample`, in this example). In this `db.ini` file, define the `PermSize` and `DatabaseCharacterSet` connection attributes.

```
vi db.ini

PermSize=200
DatabaseCharacterSet=AL32UTF8
```

4. Create the `adminUser` file in this ConfigMap directory (`cm_sample` in this example). In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

5. Create the `schema.sql` file in this ConfigMap directory (`cm_sample` in this example). In this `schema.sql` file, define the `s` sequence and the `emp` table for the `sampleuser` user. The Operator will automatically initialize your database with these object definitions.

   ```
   vi schema.sql

   create sequence sampleuser.s;
   create table sampleuser.emp (
     id number not null primary key,
     name char(32)
   );
   ```

6. Create the ConfigMap. The files in the `cm_sample` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

   In this example:

   - The name of the ConfigMap is `sample`. Replace `sample` with a name of your choosing. (`sample` is represented in **bold** in this example.)
   - This example uses `cm_sample` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_sample` with the name of your directory. (`cm_sample` is represented in **bold** in this example.)

   Use the `kubectl create` command to create the ConfigMap:

   ```
   % kubectl create configmap sample --from-file=cm_sample
   configmap/sample created
   ```

   You successfully created and deployed the `sample` ConfigMap.

7. Use the `kubectl describe` command to verify the contents of the ConfigMap. (`sample`, in this example.)

   ```
   % kubectl describe configmap sample
   Name:         sample
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>

   Data
   ====
   adminUser:
   ----
   sampleuser/samplepw

   db.ini:
   ----
   PermSize=200
   ```

```
DatabaseCharacterSet=AL32UTF8

schema.sql:
----
create sequence sampleuser.s;
create table sampleuser.emp (
  id number not null primary key,
  name char(32)
);


Events:  <none>
```

Create the TimesTenClassic object

This section creates the TimesTenClassic object. See "Defining and creating the TimesTenClassic object" and "About the TimesTenClassic object type" for detailed information on the TimesTenClassic object.

Perform these steps:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (In this example, `sample`.) The YAML file contains the definitions for the TimesTenClassic object. See "TimesTenClassicSpecSpec" for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   In this example, replace the following. (The values you can replace are represented in **bold**.)

   - `name`: Replace `sample` with the name of your TimesTenClassic object.
   - `storageClassName`: Replace `oci` with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
   - `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. Note: This example assumes a production environment and uses a value of `250Gi` for `storageSize`. For demonstration purposes, a value of `50Gi` is adequate. See the `storageSize` and the `logStorageSize` entries in "Table 15-3" for information.
   - `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location and name of your image.
   - `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.
   - `dbConfigMap`: This example uses one ConfigMap (called `sample`) for the `db.ini`, the `adminUser`, and the `schema.sql` metadata files. This ConfigMap will be included in the ProjectedVolume. This volume is mounted as `/ttconfig` in the TimesTen containers. See "Using ConfigMaps and Secrets" and "Example using one ConfigMap" for information on ConfigMaps.

   ```
   % vi sample.yaml

   apiVersion: timesten.oracle.com/v1
   kind: TimesTenClassic
   metadata:
     name: sample
   spec:
   ```

```
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - sample
```

2. Use the `kubectl create` command to create the TimesTenClassic object from the contents of the YAML file (in this example, `sample.yaml`). Doing so begins the process of deploying your active standby pair of TimesTen databases in the Kubernetes cluster.

   ```
   % kubectl create -f sample.yaml
   configmap/sample created
   timestenclassic.timesten.oracle.com/sample created
   ```

You successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your TimesTen databases begins, but is not yet complete.

Monitor deployment

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the active standby pair as it is provisioned.

> **Note:**
>
> For the `kubectl get timestenclassic` and `kubectl describe timestenclassic` commands, you can alternatively specify `kubectl get ttc` and `kubectl describe ttc` respectively. `timestenclassic` and `ttc` are synonymous when used in these commands, and return the same results. The first `kubectl get` and the first `kubectl describe` examples in this appendix use `timestenclassic`. The remaining examples in this appendix use `ttc` for simplicity.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

   ```
   % kubectl get timestenclassic sample
   NAME     STATE          ACTIVE   AGE
   sample   Initializing   None     11s
   ```

2. Use the `kubectl describe` command to view the initial provisioning in detail.

   ```
   % kubectl describe timestenclassic sample
   Name:         sample
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   API Version:  timesten.oracle.com/v1
   Kind:         TimesTenClassic
   Metadata:
     Creation Timestamp:  2021-05-31T15:35:12Z
     Generation:          1
     Resource Version:    20231755
   ```

```
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
  UID:                 517a8646-a354-11ea-a9fb-0a580aed5e4a
Spec:
  Ttspec:
    Db Config Map:
      sample
    Image:               container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250Gi
Status:
  Active Pods:       None
  High Level State:  Initializing
  Last Event:        3
  Pod Status:
    Cache Status:
      Cache Agent:        Down
      Cache UID Pwd Set:  false
      N Cache Groups:     0
    Db Status:
      Db:            Unknown
      Db Id:         0
      Db Updatable:  Unknown
    Initialized:     true
    Pod Status:
      Agent:                 Down
      Last Time Reachable:  0
      Pod IP:
      Pod Phase:             Pending
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Down
      Rep Peer P State:             Unknown
      Rep Scheme:                   Unknown
      Rep State:                    Unknown
    Times Ten Status:
      Daemon:          Down
      Instance:        Unknown
      Release:         Unknown
    Admin User File:   false
    Cache User File:   false
    Cg File:           false
    High Level State:  Down
    Intended State:    Active
    Name:              sample-0
    Schema File:       false
    Cache Status:
      Cache Agent:        Down
      Cache UID Pwd Set:  false
      N Cache Groups:     0
    Db Status:
      Db:            Unknown
      Db Id:         0
      Db Updatable:  Unknown
    Initialized:     true
    Pod Status:
      Agent:                 Down
      Last Time Reachable:  0
      Pod IP:
```

```
      Pod Phase:            Pending
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Down
      Rep Peer P State:             Unknown
      Rep Scheme:                   Unknown
      Rep State:                    Unknown
    Times Ten Status:
      Daemon:             Down
      Instance:           Unknown
      Release:            Unknown
    Admin User File:      false
    Cache User File:      false
    Cg File:              false
    High Level State:     Unknown
    Intended State:       Standby
    Name:                 sample-1
    Schema File:          false
  Rep Create Statement:   create active standby pair "sample" on
 "sample-0.sample.mynamespace.svc.cluster.local", "sample" on
 "sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store "sample" on
 "sample-0.sample.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
 store "sample" on "sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD  0
  Rep Port:               4444
  Status Version:         1.0
Events:
  Type  Reason  Age   From       Message
  ----  ------  ----  ----       -------
  -     Create  50s   ttclassic  Secret tt517a8646-a354-11ea-
a9fb-0a580aed5e4a created
  -     Create  50s   ttclassic  Service sample created
  -     Create  50s   ttclassic  StatefulSet sample created
```

3. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

```
% kubectl get ttc sample
NAME     STATE    ACTIVE     AGE
sample   Normal   sample-0   3m5s
```

4. Use the `kubectl describe` command again to view the active standby pair provisioning in detail.

   Note: In this example, the `now Normal` line displays on its own line. In the actual output, this line does not display as its own line, but at the end of the `StateChange` previous line.

```
% kubectl describe ttc sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-05-31T15:35:12Z
  Generation:          1
  Resource Version:    20232668
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
```

```
    UID:                 517a8646-a354-11ea-a9fb-0a580aed5e4a
  Spec:
    Ttspec:
      Db Config Map:
        sample
      Image:                 container-registry.oracle.com/timesten/timesten:22.1.1.9.0
      Image Pull Policy:   Always
      Image Pull Secret:   sekret
      Storage Class Name:  oci
      Storage Size:        250Gi
  Status:
    Active Pods:       sample-0
    High Level State:  Normal
    Last Event:        35
    Pod Status:
      Cache Status:
        Cache Agent:        Not Running
        Cache UID Pwd Set:  true
        N Cache Groups:     0
      Db Status:
        Db:             Loaded
        Db Id:          26
        Db Updatable:  Yes
      Initialized:     true
      Pod Status:
        Agent:                Up
        Last Time Reachable:  1590939597
        Pod IP:               192.0.2.1
        Pod Phase:            Running
      Replication Status:
        Last Time Rep State Changed:  0
        Rep Agent:                    Running
        Rep Peer P State:             start
        Rep Scheme:                   Exists
        Rep State:                    ACTIVE
      Times Ten Status:
        Daemon:          Up
        Instance:        Exists
        Release:         22.1.1.9.0
      Admin User File:   true
      Cache User File:   false
      Cg File:           false
      High Level State:  Healthy
      Intended State:    Active
      Name:              sample-0
      Schema File:       true
      Cache Status:
        Cache Agent:        Not Running
        Cache UID Pwd Set:  true
        N Cache Groups:     0
      Db Status:
        Db:             Loaded
        Db Id:          26
        Db Updatable:  No
      Initialized:     true
      Pod Status:
        Agent:                Up
        Last Time Reachable:  1590939597
        Pod IP:               192.0.2.2
        Pod Phase:            Running
      Replication Status:
```

```
      Last Time Rep State Changed:  1590939496
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Times Ten Status:
      Daemon:             Up
      Instance:           Exists
      Release:            22.1.1.9.0
    Admin User File:      true
    Cache User File:      false
    Cg File:              false
    High Level State:     Healthy
    Intended State:       Standby
    Name:                 sample-1
    Schema File:          true
  Rep Create Statement:   create active standby pair "sample" on
"sample-0.sample.mynamespace.svc.cluster.local", "sample" on
"sample-1.sample.mynamespace.svc.cluster.local" NO RETURN store "sample" on
"sample-0.sample.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
store "sample" on "sample-1.sample.mynamespace.svc.cluster.local" PORT 4444
FAILTHRESHOLD 0
  Rep Port:               4444
  Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     Create       4m43s  ttclassic  Secret tt517a8646-a354-11ea-
a9fb-0a580aed5e4a created
  -     Create       4m43s  ttclassic  Service sample created
  -     Create       4m43s  ttclassic  StatefulSet sample created
  -     StateChange  3m47s  ttclassic  Pod sample-0 Daemon Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-0 CacheAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-0 RepAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 Daemon Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 CacheAgent Unknown
  -     StateChange  3m47s  ttclassic  Pod sample-1 RepAgent Unknown
  -     StateChange  3m26s  ttclassic  Pod sample-0 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-0 Release 22.1.1.9.0
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-1 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-1 Release 22.1.1.9.0
  -     StateChange  3m26s  ttclassic  Pod sample-1 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Up
  -     StateChange  3m25s  ttclassic  Pod sample-1 Daemon Up
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  2m13s  ttclassic  Pod sample-0 Database Updatable
  -     StateChange  2m13s  ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepAgent Not Running
  -     StateChange  2m13s  ttclassic  Pod sample-0 RepScheme None
  -     StateChange  2m13s  ttclassic  Pod sample-0 Database Loaded
  -     StateChange  2m11s  ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  2m10s  ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  2m10s  ttclassic  Pod sample-0 RepState ACTIVE
  -     StateChange  113s   ttclassic  Pod sample-1 Database Loaded
  -     StateChange  113s   ttclassic  Pod sample-1 Database Not Updatable
  -     StateChange  113s   ttclassic  Pod sample-1 CacheAgent Not Running
  -     StateChange  113s   ttclassic  Pod sample-1 RepAgent Not Running
  -     StateChange  113s   ttclassic  Pod sample-1 RepScheme Exists
  -     StateChange  113s   ttclassic  Pod sample-1 RepState IDLE
  -     StateChange  106s   ttclassic  Pod sample-1 RepAgent Running
```

```
    -     StateChange  101s   ttclassic  Pod sample-1 RepState STANDBY
    -     StateChange  101s   ttclassic  TimesTenClassic was Initializing, now Normal
```

Your active standby pair of TimesTen databases are successfully deployed (as indicated by `Normal`.) There are two TimesTen databases, configured as an active standby pair. One database is active. (In this example, `sample-0` is the active database, as indicated by `Rep State ACTIVE`). The other database is standby. (In this example, `sample-1` is the standby database as indicated by `Rep State STANDBY`). The active database can be modified and queried. Changes made on the active database are replicated to the standby database. If the active database fails, the Operator automatically promotes the standby database to be the active. The formerly active database will be repaired or replaced, and will then become the standby.

Verify the existence of the underlying objects

Use the `kubectl describe` commands to verify the underlying objects.

1. StatefulSet:

```
% kubectl get statefulset sample
NAME     READY   AGE
sample   2/2     8m21s
```

2. Service:

```
% kubectl get service sample
NAME     TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
sample   ClusterIP   None         <none>        6625/TCP   9m28s
```

3. Pods:

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
sample-0                                    2/2     Running   0          10m
sample-1                                    2/2     Running   0          10m
timesten-operator-5d7dcc7948-8mnz4          1/1     Running   0          11h
```

4. PersistentVolumeClaims (PVCs):

```
% kubectl get pvc
NAME                             STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
tt-persistent-sample-0           Bound
ocid1.volume.oc1.phx.abyhqljrbxcgzyixa4pmmcwiqxgqclc7gxvdnoty367w2qn26tij6kfpx
6qq
250Gi       RWO            oci            10m
tt-persistent-sample-1           Bound
ocid1.volume.oc1.phx.abyhqljtt4qxxoj5jqiskriskh66hakaw326rbza4uigmuaezdnu53qhh
oaa
250Gi       RWO            oci            10m
```

Verify the connection to the active TimesTen database

You can run the `kubectl exec` command to invoke shells in your Pods and control TimesTen, which is running in those Pods. By default, TimesTen runs in the Pods as the `timesten` user. Once you have established a shell in the Pod, verify you can connect to the `sample` database, and that the information from the metadata files is correct. You can optionally run queries against the database or any other operations.

1. Establish a shell in the Pod.

```
% kubectl exec -it sample-0 -c tt -- /bin/bash
```

2. Connect to the `sample` database. Verify the information in the metadata files is in the database correctly. For example, attempt to connect to the database as the `sampleuser` user. Check that the `PermSize` value of `200` is correct. Check that the `sampleuser.emp` table exists.

```
 % ttIsql sample

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=sample";
Connection successful:
DSN=sample;UID=timesten;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)

Command> connect adding "uid=sampleuser;pwd=samplepw" as sampleuser;
Connection successful:
DSN=sample;UID=sampleuser;DataStore=/tt/home/timesten/datastore/sample;
DatabaseCharacterSet=AL32UTF8;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
sampleuser: Command> tables;
  SAMPLEUSER.EMP
1 table found.
```

Recover from failure

This example simulates a failure of the active TimesTen database. This is for demonstration purposes only. Do not do this in a production environment.

1. Use the `kubectl delete pod` command to delete the active database (`sample-0` in this case)

```
% kubectl delete pod sample-0
```

2. Use the `kubectl describe` command to observe how the Operator recovers from the failure. The Operator promotes the standby database (`sample-1`) to be active. Any applications that were connected to the `sample-0` database are automatically reconnected to the `sample-1` database by TimesTen. After a brief outage, the applications can continue to use the database. See "Monitoring the health of the active standby pair of databases" for information on the health and states of the active standby pair.

   Note: In this example, the text for the `Message` column displays on two lines for three state changes. However, the actual output displays on one line for each of these three state changes.

```
% kubectl describe ttc sample
Name:         sample
...
Events:
  Type  Reason       Age   From       Message
  ----  ------       ----  ----       -------
  -     StateChange  2m1s  ttclassic  TimesTenClassic sample: was Normal,
now ActiveDown
  -     StateChange  115s   ttclassic  Pod sample-1 Database Updatable: Yes
  -     StateChange  115s   ttclassic  TimesTenClassic sample:was ActiveDown, now
StandbyDown
  -     StateChange  115s   ttclassic  Pod sample-1 RepState ACTIVE
  -     StateChange  110s   ttclassic  Pod sample-0 High Level State Unknown
  -     StateChange  63s    ttclassic  Pod sample-0 Pod Phase Running
  -     StateChange  63s    ttclassic  Pod sample-0 Agent Up
  -     StateChange  63s    ttclassic  Pod sample-0 Instance Exists
  -     StateChange  63s    ttclassic  Pod sample-0 Daemon Up
  -     StateChange  63s    ttclassic  Pod sample-0 Database None
  -     StateChange  42s    ttclassic  Pod sample-0 Database Loaded
  -     StateChange  42s    ttclassic  Pod sample-0 Database Updatable: No
  -     StateChange  42s    ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  42s    ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  42s    ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  42s    ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  36s    ttclassic  Pod sample-0 High Level State Healthy
  -     StateChange  36s    ttclassic  Pod sample-0 RepState STANDBY
  -     StateChange  36s    ttclassic  TimesTenClassic sample:was StandbyDown, now
Normal
```

Kubernetes has automatically respawned a new `sample-0` Pod to replace the Pod you deleted. The Operator configured TimesTen inside of that Pod, bringing the database in the Pod up as the new standby database. The replicated pair of databases are once again functionally normally.

Cleanup

This example concludes with deleting the databases and all objects associated with TimesTenClassic. These steps are used for example purposes only. Doing these steps results in the termination of the Pods that are running the TimesTen databases as well as the deletion of the TimesTen databases themselves.

1. Delete the ConfigMap object. (`sample`, in this example.)

   ```
   % kubectl delete configmap sample
   configmap "sample" deleted
   ```

2. Delete the TimesTenClassic object and the underlying objects.

   ```
   % kubectl delete -f sample.yaml
   timestenclassic.timesten.oracle.com "sample" deleted
   ```

3. Verify the Pods that were running the TimesTen databases no longer exist.

   ```
   % kubectl get pods
   NAME                                        READY   STATUS    RESTARTS   AGE
   timesten-operator-5d7dcc7948-8mnz4          1/1     Running   0          5d23h
   ```

4. Delete the persistent storage used to hold your databases. You have to do this manually.

   ```
   % kubectl get pvc
   NAME                        STATUS   VOLUME
   CAPACITY    ACCESS MODES   STORAGECLASS   AGE
   tt-persistent-sample-0      Bound
   ...

   tt-persistent-sample-1      Bound
   ...
   % kubectl delete pvc tt-persistent-sample-0
   ```

```
persistentvolumeclaim "tt-persistent-sample-0" deleted
% kubectl delete pvc tt-persistent-sample-1
persistentvolumeclaim "tt-persistent-sample-1" deleted
```

5. If you no longer want to run the Operator, you can stop it. Navigate to the `/deploy` directory (*`kube_files`*`/deploy`, in this example) and use the `kubectl delete` command to stop the operator.

```
% cd kube_files/deploy
% kubectl delete -f operator.yaml
deployment.apps "timesten-operator" deleted
```

B
TimesTen Cache in TimesTen Classic Example

This appendix provides a working example for using TimesTen Cache with active standby pair of TimesTen databases in your Kubernetes environment. This example should not be used for production purposes. It assumes a test environment. Your Oracle Database should be customized with the settings specific to your environment.

Topics:

Setting up the Oracle Database to cache data

The following sections describe the tasks that must be performed in the Oracle Database:

Create the Oracle Database users

Before you can use TimesTen Cache, you must create the following users in your Oracle database:

- A cache administration user. This user creates and maintains Oracle Database objects that store information about the cache environment. This user also enforces predefined behaviors of cache group types.
- One or more schema users who owns Oracle Database tables that are cached in a TimesTen database.

See "Create the Oracle database users and default tablespace" in the *Oracle TimesTen In-Memory Database Cache Guide* for information.

This example creates the `cacheuser2` cache administration user and the `oratt` schema user in the Oracle Database.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the `sys` user. Then, create a default tablespace to

store the TimesTen Cache management objects. See "Create the Oracle database users and default tablespace" in the *Oracle TimesTen In-Memory Database Cache Guide* for information.

This example creates the `cachetablespace2` tablespace.

```
% sqlplus sys/syspwd@oracache as sysdba

SQL> CREATE TABLESPACE cachetablespace2 DATAFILE 'datatt2.dbf' SIZE 100M;

Tablespace created.
```

2. Use SQL*Plus to create the schema user. Grant this schema user the minimum privileges required to create tables in the Oracle Database to be cached in your TimesTen database.

   This example creates the `oratt` schema user.

   ```
   SQL> CREATE USER oratt IDENTIFIED BY oraclepwd;

   User created.

   SQL> GRANT CREATE SESSION, RESOURCE TO oratt;

   Grant succeeded.
   ```

3. Use SQL*Plus to create the cache administration user. Assign the `cachetablespace2` tablespace to this user. You will later use the same name of this Oracle cache administration user in the `cacheUser` metadata file.

   This example creates the `cacheuser2` user.

   ```
   SQL> CREATE USER cacheuser2 IDENTIFIED BY oraclepwd
          DEFAULT TABLESPACE cachetablespace2
          QUOTA UNLIMITED ON cachetablespace2;

   User created.

   SQL> commit;

   Commit complete.

   SQL> exit
   ```

Grant privileges to the cache administration user

The cache administration user must be granted a specific set of privileges depending on the cache group types that will be created in the TimesTen databases and the operations performed on those cache groups. TimesTen provides the `grantCacheAdminPrivileges.sql` SQL*Plus script that you can run in your Oracle Database to grant the cache administration user the minimum set of privileges required to perform cache operations. See "Grant privileges to the Oracle database users" and see "Required privileges for cache operations" in the *Oracle TimesTen In-Memory Database Cache Guide* for more information on these privileges.

Perform these steps to run the `grantCacheAdminPrivileges.sql` script:

1. Create a shell from which you can access your Oracle Database, and then from the directory of your choice, create an empty subdirectory. This example creates the `oraclescripts` subdirectory.

```
% mkdir -p oraclescripts
```

2. From your Linux development host, use the `kubectl cp` command to copy the `grantCacheAdminPrivileges.sql` script from the *`installation_dir`*`/oraclescripts` directory on your Linux development host to the `oraclescripts` directory that you just created. Recall that the *`installation_dir`* directory was created when you unpacked the TimesTen distribution. See "Unpack the TimesTen and the TimesTen Kubernetes Operator distributions" for information on unpacking the TimesTen distribution.

   ```
   % cp /installation_dir/oraclescripts/grantCacheAdminPrivileges.sql
   database-oracle:oraclescripts/grantCacheAdminPrivileges.sql
   ```

3. From your shell, verify the script is located in the `oraclescripts` directory.

   ```
   % ls oraclescripts
   grantCacheAdminPrivileges.sql
   ```

4. Use SQL*Plus to connect to the Oracle Database as the `sys` user. Then, run the *`oraclescripts`*`/grantCacheAdminPrivileges.sql` script. This script grants the `cacheuser2` cache administration user the minimum set of privileges required to perform cache group operations. See "Grant privileges to the Oracle database users" in the *Oracle TimesTen In-Memory Database Cache Guide* for more information.

   ```
   % sqlplus sys/syspwd@oracache as sysdba

   SQL> @grantCacheAdminPrivileges "cacheuser2";

   Please enter the administrator user id
   The value chosen for administrator user id is cacheuser2

   TT_CACHE_ADMIN_ROLE role already exists
   ***************** Initialization for cache admin begins ******************
   0. Granting the CREATE SESSION privilege to CACHEUSER2
   1. Granting the TT_CACHE_ADMIN_ROLE to CACHEUSER2
   2. Granting the DBMS_LOCK package privilege to CACHEUSER2
   3. Granting the DBMS_DDL package privilege to CACHEUSER2
   4. Granting the DBMS_FLASHBACK package privilege to CACHEUSER2
   5. Granting the CREATE SEQUENCE privilege to CACHEUSER2
   6. Granting the CREATE CLUSTER privilege to CACHEUSER2
   7. Granting the CREATE OPERATOR privilege to CACHEUSER2
   8. Granting the CREATE INDEXTYPE privilege to CACHEUSER2
   9. Granting the CREATE TABLE privilege to CACHEUSER2
   10. Granting the CREATE PROCEDURE  privilege to CACHEUSER2
   11. Granting the CREATE ANY TRIGGER  privilege to CACHEUSER2
   12. Granting the GRANT UNLIMITED TABLESPACE privilege to CACHEUSER2
   13. Granting the DBMS_LOB package privilege to CACHEUSER2
   14. Granting the SELECT on SYS.ALL_OBJECTS privilege to CACHEUSER2
   15. Granting the SELECT on SYS.ALL_SYNONYMS privilege to CACHEUSER2
   16. Checking if the cache administrator user has permissions on the
       default tablespace
       Permission exists
   18. Granting the CREATE TYPE privilege to CACHEUSER2
   19. Granting the SELECT on SYS.GV$LOCK privilege to CACHEUSER2
   20. Granting the SELECT on SYS.GV$SESSION privilege  to CACHEUSER2
   21. Granting the SELECT on SYS.DBA_DATA_FILES privilege  to CACHEUSER2
   22. Granting the SELECT on SYS.USER_USERS privilege  to CACHEUSER2
   23. Granting the SELECT on SYS.USER_FREE_SPACE privilege  to CACHEUSER2
   24. Granting the SELECT on SYS.USER_TS_QUOTAS privilege  to CACHEUSER2
   25. Granting the SELECT on SYS.USER_SYS_PRIVS privilege  to CACHEUSER2
   26. Granting the SELECT on SYS.V$DATABASE privilege  to CACHEUSER2 (optional)
   27. Granting the SELECT ANY TRANSACTION privilege to CACHEUSER2
   ********* Initialization for cache admin user done successfully *********
   ```

You have successfully run the `grantCacheAdminPrivileges.sql` script in the Oracle Database.

Create the Oracle Database tables to be cached

This example creates two tables in the `oratt` user schema. See "Create the Oracle Database users" for information on this user.

- `readtab`: This table will later be cached in a read-only cache group.
- `writetab`: This table will later be cached in an AWT cache group.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the `sys` user. Then create the `oratt.readtab` and the `oratt.writetab` tables.

```
% sqlplus sys/syspwd@oracache as sysdba

SQL> CREATE TABLE oratt.readtab (keyval NUMBER NOT NULL PRIMARY KEY,
       str VARCHAR2(32));

Table created.

SQL> CREATE TABLE oratt.writetab (pk NUMBER NOT NULL PRIMARY KEY,
       attr VARCHAR2(40));

Table created.
```

2. Use SQL*Plus to insert rows into the `oratt.readtab` and the `oratt.writetab` tables. Then verify the rows have been inserted.

```
SQL> INSERT INTO oratt.readtab VALUES (1,'Hello');

1 row created.

SQL> INSERT INTO oratt.readtab VALUES (2,'World');

1 row created.

SQL> INSERT INTO oratt.writetab VALUES (100, 'TimesTen');

1 row created.

SQL> INSERT INTO oratt.writetab VALUES (101, 'Cache');

1 row created.

SQL> commit;

Commit complete.
```

Verify the rows have been inserted into the tables.

```
SQL> SELECT * FROM oratt.readtab;

    KEYVAL STR
---------- --------------------------------
         1 Hello
         2 World

SQL>  SELECT * FROM oratt.writetab;
```

```
        PK ATTR
---------- ----------------------------------------
       100 TimesTen
       101 Cache
```

3. Use SQL*Plus to grant the `SELECT` privilege on the `oratt.readtab` table and the `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the `oratt.writetab` table to the cache administration user (`cacheuser2`, in this example).

   ```
   SQL> GRANT SELECT ON oratt.readtab TO cacheuser2;

   Grant succeeded.

   SQL> GRANT SELECT ON oratt.writetab TO cacheuser2;

   Grant succeeded.

   SQL> GRANT INSERT ON oratt.writetab TO cacheuser2;

   Grant succeeded.

   SQL> GRANT UPDATE ON oratt.writetab TO cacheuser2;

   Grant succeeded.

   SQL> GRANT DELETE ON oratt.writetab TO cacheuser2;

   Grant succeeded.
   ```

4. Use SQL*Plus to query the `nls_database_parameters` system view to determine the Oracle Database database character set. The Oracle Database database character set must match the TimesTen database character set. (The TimesTen database character set will be set later. See "Creating the metadata files and the Kubernetes facility" for details.)

   In this example, the query returns the `AL32UTF8` database character set.

   ```
   SQL> SELECT value FROM nls_database_parameters WHERE
          parameter='NLS_CHARACTERSET';

   VALUE
   ------------------------------------------------------------------------------
   AL32UTF8
   ```

You have successfully created the Oracle Database tables that will be cached in the TimesTen cache group tables.

Creating the metadata files and the Kubernetes facility

There are metadata files that are specific to using TimesTen Cache:

- `cacheUser`: This file is required. The user in this file is created in the TimesTen databases and serves as the cache manager. The name of this user must match the name of the cache administration user that you created in the Oracle Database. See "Create the Oracle Database users" for information on the cache administration user in the Oracle Database. Also see "cacheUser" for more information on the `cacheUser` metadata file.
- `cachegroups.sql`: This file is required. The contents of this file contain the `CREATE CACHE GROUP` definitions. The file can also contain the `LOAD CACHE GROUP` statement and the built-in procedures to update statistics on the cache group tables (such as,

ttOptEstimateStats and ttOptUpdateStats). See "cachegroups.sql" for more information on this file.

- tnsnames.ora: This file is required. It defines Oracle Net Services to which applications connect. For TimesTen Cache, this file configures the connectivity between TimesTen and the Oracle Database (from which data is being cached). In this context, TimesTen is the application that is the connection to the Oracle Database. See "tnsnames.ora" for more information on this file.
- sqlnet.ora: This file may be required. It may be necessary depending on your Oracle Database configuration. The file defines options for how client applications communicate with the Oracle Database. In this context, TimesTen is the application. The tnsnames.ora and sqlnet.ora files together define how an application communicates with the Oracle Database. See "sqlnet.ora" for information on this file.
- db.ini: This file is required if you are using TimesTen Cache. The contents of this file contain TimesTen connection attributes for your TimesTen databases, which will be included in TimesTen's sys.odbc.ini file. For TimesTen Cache, you must specify the OracleNetServiceName and the DatabaseCharacterSet connection attributes. The DatabaseCharacterSet connection attribute must match the Oracle database character set. See "db.ini" for more information on this file.
- schema.sql: The contents of this file contain database objects, such as tables, sequences, and users. The instance administrator uses the ttIsql utility to run this file immediately after the database is created. This file is run before the Operator configures TimesTen Cache or replication, so ensure there are no cache definitions in this file.

  In TimesTen Cache, one or more cache table users own the cache tables. If this cache table user is not the cache manager user, then you must specify the schema.sql file and in it you must include the schema user and assign the appropriate privileges to this schema user. For example, if the oratt schema user was created in the Oracle Database, and this user is not the TimesTen cache manager user, you must create the TimesTen oratt user in this file. See "Create the Oracle Database users" for more information on the schema users in the Oracle Database. Also see "schema.sql" for more information on the schema.sql file.

In addition, you can use these other supported metadata files:

- adminUser: The user in this file is created in the TimesTen databases and is granted ADMIN privileges. See "adminUser" for more information on this file.
- epilog.sql: The contents of this file contain operations that must be performed after the Operator configures replication. For example, if you are using XLA, you could create replicated bookmarks for XLA in this file. This file is run after cache and replication have been configured. See "epilog.sql" for more information on this file.

You can include these metadata files in one or more Kubernetes facilities (for example, in a Kubernetes Secret, in a ConfigMap, or in an init container). This ensures the metadata files are populated in the /ttconfig directory of the TimesTen containers. Note that there is no requirement as to how to get the metadata files into this /ttconfig directory. See "Populating the /ttconfig directory" for more information.

This example uses the ConfigMap facility to populate the /ttconfig directory in your TimesTen containers. The adminUser, db.ini, schema.sql, cacheUser, `cachegroups.sql`, `tnsnames.ora`, and `sqlnet.ora` metadata files are used in this example.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_cachetest` subdirectory. (The `cm_cachetest` directory is used in the remainder of this example to denote this directory.)

   ```
   % mkdir -p cm_cachetest
   ```

2. Navigate to the ConfigMap directory.

   ```
   % cd cm_cachetest
   ```

3. Create the `adminUser` file in this ConfigMap directory (`cm_cachetest`, in this example). In this `adminUser` file, create the `sampleuser` user with the `samplepw` password.

   ```
   vi adminUser

   sampleuser/samplepw
   ```

4. Create the `db.ini` file in this ConfigMap directory (`cm_cachetest`, in this example). In this `db.ini` file, define the `PermSize`, `DatabaseCharacterSet`, and the `OracleNetServiceName` connection attributes. The `DatabaseCharacterSet` value must match the database character set value in the Oracle Database. See "Create the Oracle Database tables to be cached" for information on how to query the `nls_database_parameters` system view to determine the Oracle Database database character set. In this example, the value is `AL32UTF8`.

   ```
   vi db.ini

   PermSize=200
   DatabaseCharacterSet=AL32UTF8
   OracleNetServiceName=Oracache
   ```

5. Create the `schema.sql` file in this ConfigMap directory (`cm_cachetest`, in this example). In this example, create the `oratt` user. Recall that this user was previously created in the Oracle Database. See "Create the Oracle Database users" for information on the `oratt` user in the Oracle Database.

   ```
   vi schema.sql

   create user oratt identified by ttpwd;
   grant admin to oratt;
   ```

6. Create the `cacheUser` metadata file in this ConfigMap directory (`cm_cachetest`, in this example). The `cacheUser` file must contain one line of the form `cacheuser/ttpassword/orapassword`, where `cacheuser` is the user you wish to designate as the cache manager in the TimesTen database, `ttpassword` is the TimesTen password you wish to assign to this user, and `orapassword` is the Oracle Database password that has already been assigned to the Oracle Database cache administration user. Note that the `cacheUser` name in this file must match the Oracle Database cache administration user that you previously created. See "Create the Oracle Database users" for more information on the Oracle Database cache administration user.

   In this example, the `cacheuser2` user with password of `oraclepwd` was already created in the Oracle Database. Therefore, supply `cacheuser2` as the TimesTen cache manager user. You can assign any TimesTen password to this TimesTen cache manager user. This example assigns `ttpwd`.

```
vi cacheuser

cacheuser2/ttpwd/oraclepwd
```

7. Create the `cachegroups.sql` metadata file in this ConfigMap directory (`cm_cachetest`, in this example). The `cachegroups.sql` file contains the cache group definitions. In this example, a dynamic AWT cache group and a read-only cache group are created. In addition, the `LOAD CACHE GROUP` statement is included to load rows from the `oratt.readtab` cached table in the Oracle Database into the `oratt.readtab` cache table in the TimesTen database.

   ```
   vi cachegroups.sql

   CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP writecache
   FROM oratt.writetab (pk NUMBER NOT NULL PRIMARY KEY,attr VARCHAR2(40));

   CREATE READONLY CACHE GROUP readcache
   AUTOREFRESH
     INTERVAL 5 SECONDS
   FROM oratt.readtab (keyval NUMBER NOT NULL PRIMARY KEY,str VARCHAR2(32));

   LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
   ```

8. Create the `tnsnames.ora` metadata file in this ConfigMap directory (`cm_cachetest`, in this example).

   ```
   vi tnsnames.ora

   OraTest =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
        (PORT = 1521))
      (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = OraTest.my.sample.com)))
   OraCache =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
        (PORT = 1521))
      (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = OraCache.my.sample.com)))
   ```

9. Create the `sqlnet.ora` metadata file in this ConfigMap directory (`cm_cachetest`, in this example).

   ```
   vi sqlnet.ora

   NAME.DIRECTORY_PATH= {TNSNAMES, EZCONNECT, HOSTNAME}
   SQLNET.EXPIRE_TIME = 10
   SSL_VERSION = 1.2
   ```

10. Use the Linux `ls` command to verify the metadata files are in the ConfigMap directory (`cm_cachetest`, in this example).

    ```
    % ls
    adminUser        cacheUser  schema.sql  tnsnames.ora
    cachegroups.sql  db.ini     sqlnet.ora
    ```

11. Create the ConfigMap. The files in the `cm_cachetest` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

    In this example:

- The name of the ConfigMap is `cachetest`. Replace `cachetest` with a name of your choosing. (`cachetest` is represented in **bold** in this example.)
- This example uses `cm_cachetest` as the directory where the files that will be copied into the ConfigMap reside. If you use a different directory, replace `cm_cachetest` with the name of your directory. (`cm_cachetest` is represented in **bold** in this example.)

Use the `kubectl create` command to create the ConfigMap:

```
% kubectl create configmap cachetest --from-file=cm_cachetest
configmap/cachetest created
```

12. Use the `kubectl describe` command to verify the contents of the ConfigMap. (`cachetest`, in this example.) The metadata files are represented in **bold**.

```
% kubectl describe configmap cachetest;
Name:         cachetest
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
tnsnames.ora:
----

OraTest =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraTest.my.sample.com)))
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.sample.com)))

adminUser:
----
sampleuser/samplepw

cacheUser:
----
cacheuser2/ttpwd/oraclepwd

cachegroups.sql:
----
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP writecache
FROM oratt.writetab (
  pk NUMBER NOT NULL PRIMARY KEY,
  attr VARCHAR2(40)
);

CREATE READONLY CACHE GROUP readcache
AUTOREFRESH
  INTERVAL 5 SECONDS
FROM oratt.readtab (
  keyval NUMBER NOT NULL PRIMARY KEY,
  str VARCHAR2(32)
);

LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;

db.ini:
----
permSize=200
databaseCharacterSet=AL32UTF8
oracleNetServiceName=Oracache

schema.sql:
----
create user oratt identified by ttpwd;
grant admin to oratt;

sqlnet.ora:
----
NAME.DIRECTORY_PATH= {TNSNAMES, EZCONNECT, HOSTNAME}
SQLNET.EXPIRE_TIME = 10
SSL_VERSION = 1.2


Events:  <none>
```

You have successfully created and deployed the `cachetest` ConfigMap.

Creating the TimesTenClassic object

This section creates the TimesTenClassic object. See "Defining and creating the TimesTenClassic object" and "About the TimesTenClassic object type" for detailed information on the TimesTenClassic object.

Perform these steps:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenClassic object. (In this example, `cachetest`.) The YAML file contains the definitions for the TimesTenClassic object. See "TimesTenClassicSpecSpec" for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   In this example, note these fields:

   - `name`: Replace `cachetest` with the name of your TimesTenClassic object (represented in **bold**).
   - `storageClassName`: Replace `oci` with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
   - `storageSize`: Replace `250Gi` with the amount of storage that should be requested for each Pod to hold TimesTen. Note: This example assumes a production environment and uses a value of `250Gi` for `storageSize`. For demonstration purposes, a value of `50Gi` is adequate.
   - `image`: Replace `container-registry.oracle.com/timesten/timesten:22.1.1.9.0` with the location and name of your image.
   - `imagePullSecret`: Replace `sekret` with the image pull secret that Kubernetes should use to fetch the TimesTen image.

- `dbConfigMap`: This example uses one ConfigMap (called `cachetest`) for the metadata files (represented in **bold**).

```
% vi cachetest.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: cachetest
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - cachetest
```

2. Use the `kubectl create` command to create the TimesTenClassic object from the contents of the YAML file (in this example, `cachetest.yaml`). Doing so begins the process of deploying your active standby pair of TimesTen databases in the Kubernetes cluster.

```
% kubectl create -f cachetest.yaml
timestenclassic.timesten.oracle.com/cachetest created
```

You have successfully created the TimesTenClassic object in the Kubernetes cluster. The process of deploying your TimesTen databases begins, but is not yet complete.

Monitoring the deployment of the TimesTenClassic object

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the active standby pair as it is provisioned.

1. Use the `kubectl get` command and review the `STATE` field. Observe the value is `Initializing`. The active standby pair provisioning has begun, but is not yet complete.

```
% kubectl get ttc cachetest
NAME         STATE          ACTIVE    AGE
cachetest    Initializing   None      41s
```

2. Use the `kubectl get` command again to see if value of the `STATE` field has changed. In this example, the value is `Normal`, indicating the active standby pair of databases are now provisioned and the process is complete.

```
% kubectl get ttc cachetest
NAME         STATE     ACTIVE         AGE
cachetest    Normal    cachetest-0    3m58s
```

3. Use the `kubectl describe` command to view the active standby pair provisioning in detail.

   Note the following:

   - The `cachetest` Configmap has been correctly referenced in the `dbConfigMap` field (represented in **bold**).
   - The cache agent is running in the active and the standby Pods (represented in **bold**).
   - The cache administration user UID and password have been set in the active and the standby Pods (represented in **bold**).

- Two cache groups have been created in the active and the standby Pods (represented in **bold**).
- The replication agent is running in the active and standby Pods (represented in **bold**).

```
% kubectl describe ttc cachetest
Name:         cachetest
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2022-10-24T03:29:48Z
  Generation:          1
  Resource Version:    78390500
  Self Link:           /apis/timesten.oracle.com/v1/namespaces/mynamespace/
timestenclassics/cachetest
  UID:                 2b18d81d-15a9-11eb-b999-be712d29a81e
Spec:
  Ttspec:
    Db Config Map:
      cachetest
    Image:               container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250Gi
Status:
  Active Pods:       cachetest-0
  High Level State:  Normal
  Last Event:        28
  Pod Status:
    Cache Status:
      Cache Agent:        Running
      Cache UID Pwd Set:  true
      N Cache Groups:     2
    Db Status:
      Db:             Loaded
      Db Id:          30
      Db Updatable:   Yes
    Initialized:      true
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1603510527
      Pod IP:               10.244.7.92
      Pod Phase:            Running
    Replication Status:
      Last Time Rep State Changed:  0
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    ACTIVE
    Times Ten Status:
      Daemon:          Up
      Instance:        Exists
      Release:         22.1.1.9.0
    Admin User File:   true
    Cache User File:   true
    Cg File:           true
    High Level State:  Healthy
    Intended State:    Active
    Name:              cachetest-0
    Schema File:       true
    Cache Status:
      Cache Agent:        Running
      Cache UID Pwd Set:  true
      N Cache Groups:     2
    Db Status:
      Db:            Loaded
      Db Id:         30
      Db Updatable:  No
    Initialized:     true
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1603510527
      Pod IP:               10.244.8.170
      Pod Phase:            Running
    Replication Status:
      Last Time Rep State Changed:  1603510411
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Times Ten Status:
      Daemon:            Up
      Instance:          Exists
      Release:           22.1.1.9.0
    Admin User File:     true
    Cache User File:     true
    Cg File:             true
    High Level State:    Healthy
    Intended State:      Standby
    Name:                cachetest-1
    Schema File:         true
  Rep Create Statement:  create active standby pair "cachetest" on
  "cachetest-0.cachetest.mynamespace.svc.cluster.local", "cachetest" on
  "cachetest-1.cachetest.mynamespace.svc.cluster.local" NO RETURN store
  "cachetest" on "cachetest-0.cachetest.mynamespace.svc.cluster.local"
  PORT 4444 FAILTHRESHOLD 0 store "cachetest" on
 "cachetest-1.cachetest.mynamespace.svc.cluster.local" PORT 4444 FAILTHRESHOLD 0
  Rep Port:              4444
  Status Version:        1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     Create       5m40s  ttclassic  Secret tt2b18d81d-15a9-11eb-b999-
be712d29a81e created
  -     Create       5m40s  ttclassic  Service cachetest created
  -     Create       5m40s  ttclassic  StatefulSet cachetest created
  -     StateChange  4m28s  ttclassic  Pod cachetest-0 Agent Up
  -     StateChange  4m28s  ttclassic  Pod cachetest-0 Release 22.1.1.9.0
  -     StateChange  4m28s  ttclassic  Pod cachetest-0 Daemon Up
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 RepScheme None
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 RepAgent Not Running
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 RepState IDLE
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 Database Loaded
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 Database Updatable
  -     StateChange  3m18s  ttclassic  Pod cachetest-0 CacheAgent Not Running
  -     StateChange  2m57s  ttclassic  Pod cachetest-0 CacheAgent Running
  -     StateChange  2m47s  ttclassic  Pod cachetest-1 Agent Up
  -     StateChange  2m47s  ttclassic  Pod cachetest-1 Release 22.1.1.9.0
  -     StateChange  2m46s  ttclassic  Pod cachetest-0 RepAgent Running
  -     StateChange  2m46s  ttclassic  Pod cachetest-0 RepScheme Exists
  -     StateChange  2m46s  ttclassic  Pod cachetest-0 RepState ACTIVE
  -     StateChange  2m46s  ttclassic  Pod cachetest-1 Daemon Up
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 CacheAgent Running
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 Database Not Updatable
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 Database Loaded
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 RepAgent Not Running
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 RepScheme Exists
  -     StateChange  2m9s   ttclassic  Pod cachetest-1 RepState IDLE
  -     StateChange  2m3s   ttclassic  Pod cachetest-1 RepAgent Running
  -     StateChange  118s   ttclassic  Pod cachetest-1 RepState STANDBY
  -     StateChange  118s   ttclassic  TimesTenClassic was Initializing, now
Normal
```

Your active standby pair of TimesTen databases are successfully deployed (as indicated by `Normal`.) You are now ready to verify that TimesTen Cache is configured correctly and is working properly.

Verifying that TimesTen Cache is configured correctly

To verify that TimesTen Cache is configured correctly and is working properly, perform the following steps:

1. Review the active (`cachetest-0`, in this example) Pod and the standby Pod (`cachetest-1`, in this example).

   ```
   % kubectl get pods
   NAME                                       READY   STATUS    RESTARTS   AGE
   cachetest-0                                2/2     Running   0          8m16s
   cachetest-1                                2/2     Running   0          8m15s
   timesten-operator-f84766548-tch7s          1/1     Running   0          36d
   ```

2. Use the `kubectl exec -it` command to invoke the shell in the active Pod (`cachetest-0`, in this example).

   ```
   % kubectl exec -it cachetest-0 -c tt -- /bin/bash
   ```

3. Use `ttIsql` to connect to the `cachetest` database. Confirm the TimesTen connection attributes are correct. In particular, note that the `OracleNetServiceName` connection attribute is correctly set to `Oracache` (represented in **bold**).

   ```
   % ttIsql cachetest;

   Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=cachetest";
   Connection successful: DSN=cachetest;UID=timesten;DataStore=/tt/home/
   timesten/datastore/cachetest;
   DatabaseCharacterSet=AL32UTF8;AutoCreate=0;
   PermSize=200;OracleNetServiceName=Oracache;DDLReplicationLevel=3;
   ForceDisconnectEnabled=1;
   (Default setting AutoCommit=1)
   ```

4. Use the `ttIsql cachegroups` to view the definition of the `cacheuser2.readcache` and the `cacheuser2.writecache` cache groups.

```
Command> cachegroups;

Cache Group CACHEUSER2.READCACHE:

  Cache Group Type: Read Only
  Autorefresh: Yes
  Autorefresh Mode: Incremental
  Autorefresh State: On
  Autorefresh Interval: 5 Seconds
  Autorefresh Status: ok
  Aging: No aging defined

  Root Table: ORATT.READTAB
  Table Type: Read Only

Cache Group CACHEUSER2.WRITECACHE:

  Cache Group Type: Asynchronous Writethrough (Dynamic)
  Autorefresh: No
  Aging: LRU on

  Root Table: ORATT.WRITETAB
  Table Type: Propagate

2 cache groups found.
```

5. Use `ttIsql` to query the `oratt.readtab` cache table. Note that the data from the `oratt.readtab` cached table in the Oracle Database is correctly loaded in the `oratt.readcache` cache table in the TimesTen database. Recall that you specified the `LOAD CACHE GROUP` statement in the `cachegroups.sql` metadata file. See Creating the metadata files and the Kubernetes facility for information on this `cachegroups.sql` metadata file.

```
Command> SELECT * FROM oratt.readtab;
< 1, Hello >
< 2, World >
2 rows found.
```

You have verified that the cache groups were created and data was correctly loaded in the `oratt.readtab` table.

Performing operations on the cache group tables

The examples in this section perform operations on the `oratt.readtab` and the `oratt.writetab` tables to verify that TimesTen Cache is working properly.

- Perform operations on the oratt.readtab table
- Perform operations on the oratt.writetab table

Perform operations on the oratt.readtab table

This section performs operations on the `oratt.readtab` table.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the schema user (`oratt`, in this example). Then, insert a new row, delete an existing row, and update an existing row in the `oratt.readtab` table of the Oracle Database and commit the changes.

```
% sqlplus oratt/oraclepwd@oracache;

SQL> INSERT INTO oratt.readtab VALUES (3,'Welcome');

1 row created.

SQL> DELETE FROM oratt.readtab WHERE keyval=2;

1 row deleted.

SQL> UPDATE oratt.readtab SET str='Hi' WHERE keyval=1;

1 row updated.

SQL> COMMIT;

Commit complete.
```

Since the read-only cache group was created with an autorefresh interval of 5 seconds, the TimesTen `oratt.readtab` cache table in the `readcache` cache group is automatically refreshed after 5 seconds with the committed updates from the cached `oratt.readtab` table of the Oracle Database. The next step is to test that the data was correctly propagated from the Oracle Database to the TimesTen database.

2. Use the `kubectl exec -it` command to invoke the shell in the container of the Pod that is running the TimesTen active database (`cachetest-0`, in this example).

```
% kubectl exec -it cachetest-0 -c tt -- /bin/bash
```

3. Use the TimesTen `ttIsql` utility to connect to the `cachetest` database. Query the TimesTen `oratt.readtab` table to verify that the table has been updated with the committed updates from the Oracle Database.

```
% ttIsql cachetest;

Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=cachetest";
Connection successful: DSN=cachetest;UID=timesten;DataStore=/tt/home/
timesten/datastore/cachetest;
DatabaseCharacterSet=AL32UTF8;AutoCreate=0;
PermSize=200;OracleNetServiceName=Oracache;DDLReplicationLevel=3;
ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)

Command> SELECT * FROM oratt.readtab;
< 1, Hi >
< 3, Welcome >
2 rows found.
```

You have verified that TimesTen Cache is working correctly for the `oratt.readtab` table and the `readcache` cachegroup.

Perform operations on the oratt.writetab table

This example performs operations on the `oratt.writetab` table.

1. Use the `kubectl exec -it` command to invoke the shell in the container of the Pod that is running the TimesTen active database (`cachetest-0`, in this example).

   ```
   % kubectl exec -it cachetest-0 -c tt -- /bin/bash
   ```

2. Use the TimesTen `ttIsql` utility to connect to the `cachetest` database as the cache manager user (`cacheuser2`, in this example). Issue a `SELECT` statement on the TimesTen `oratt.writetab` table. Recall that the `writecache` cache group is a dynamic cache group. Thus by issuing the `SELECT` statement, the cache instance is automatically loaded from the cached Oracle Database table, if the data is not found in the TimeTen cache table.

   ```
   % ttIsql "DSN=cachetest;UID=cacheuser2;PWD=ttpwd;OraclePWD=oraclepwd";

   Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=cachetest;UID=cacheuser2;PWD=********;OraclePWD=********";
   Connection successful: DSN=cachetest;UID=cacheuser2;DataStore=/tt/home/timesten/
   datastore/cachetest;
   DatabaseCharacterSet=AL32UTF8;AutoCreate=0;
   PermSize=200;OracleNetServiceName=Oracache;DDLReplicationLevel=3;
   ForceDisconnectEnabled=1;
   (Default setting AutoCommit=1)

   Command> SELECT * FROM oratt.writetab WHERE pk=100;
   < 100, TimesTen >
   1 row found.
   ```

3. Use `ttIsql` to insert a new row, delete an existing row, and update an existing row in the TimesTen `oratt.writetab` cache table, and commit the changes.

   ```
   Command> INSERT INTO oratt.writetab VALUES (102,'Cache');
   1 row inserted.
   Command> DELETE FROM oratt.writetab WHERE pk=101;
   1 row deleted.
   Command> UPDATE oratt.writetab SET attr='Oracle' WHERE pk=100;
   1 row updated.
   Command> COMMIT;
   ```

   The committed updates on the TimesTen `oratt.writetab` cache table in the `writecache` cache group should automatically be propagated to the `oratt.writetab` table in the Oracle Database.

4. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle database as the schema user (`oratt`, in this example). Then query the contents of the `oratt.writetab` table in the Oracle Database to verify the committed updates from the TimesTen database have been propagated to the `oratt.writetab` table of the Oracle Database.

   ```
   % sqlplus oratt/oraclepwd@orapcache;

   SQL> SELECT * FROM oratt.writetab ORDER BY pk;

           PK ATTR
   ---------- ----------------------------------------
          100 Oracle
          102 Cache
   ```

You have verified that TimesTen Cache is working correctly for the `oratt.writetab` table and the `writecache` cachegroup.

Cleaning up the cache metadata on the Oracle Database

When you create certain types of cache groups in a TimesTen database, TimesTen stores metadata about that cache group in the Oracle Database. If you later delete that TimesTen database, TimesTen does not automatically delete the metadata in the Oracle Database. As a result, metadata can accumulate on the Oracle Database. See "Dropping Oracle Database objects used by cache groups with autorefresh" in the *Oracle TimesTen In-Memory Database Cache Guide* for more information.

However, in a Kubernetes environment, if you provide a `cacheUser` metadata file and a `cachegroups.sql` metadata file when you initially create the TimesTenClassic object, then, by default, the Operator automatically cleans up the Oracle Database metadata if you delete that TimesTenClassic object.

If you do not want the Operator to automatically clean up the Oracle Database, you set the `cacheCleanup` field in the TimesTenClassic object definition to `false`. See the `cacheCleanup` entry in "Table 15-3" for more information. Also see "Configuration metadata details" for information on the `cacheUser` and the `cachegroups.sql` files.

C

TimesTen Cache in TimesTen Scaleout Example

The TimesTen Operator supports the use of TimesTen Cache with TimesTen Scaleout in your Kubernetes environment.

This appendix shows a complete example. The example should not be used for production purposes. It assumes a test environment. Your Oracle Database should be customized with the settings specific to your environment.

Topics:

- Set up the Oracle Database
- Create the metadata files and the Kubernetes facility
- Create the TimesTenScaleout object
- Monitor the deployment
- Verify TimesTen Cache is configured correctly
- Perform operations on the oratt_grid.readtab table
- Clean up the cache metadata on the Oracle Database

Set up the Oracle Database

You need to complete tasks in the Oracle database before using TimesTen Cache. The tasks are described in the following sections:

- Create the Oracle Database users
- Grant privileges to the cache administration user
- Create the Oracle Database table to be cached

Create the Oracle Database users

Before you can use TimesTen Cache, you need to create the following users in your Oracle database:

- A cache administration user. This user creates and maintains Oracle Database objects that store information about the cache environment. This user also enforces predefined behaviors of cache group types.
- One or more schema users who owns Oracle Database tables that are cached in a TimesTen database.

See Create the Oracle database users and default tablespace in the *Oracle TimesTen In-Memory Database Cache Guide*.

This example creates the `cacheuser_grid` cache administration user and the `oratt_grid` schema user in the Oracle Database.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the `sys` user. Then, create a default tablespace for the schema user and a default tablespace for the cache admininstration user.

   ```
   sqlplus sys/syspwd@oracache as sysdba
   ```

   Create a tablespace for the schema user (`oratt_grid`).

   ```
   CREATE TABLESPACE cachegridtablespace1 DATAFILE 'datattgrid1.dbf'
   SIZE 100M;
   ```

   The output is the following.

   ```
   Tablespace created.
   ```

   Create a tablespace for the cache administration user (`cacheuser_grid`).

   ```
   CREATE TABLESPACE cachegridtablespace2 DATAFILE 'datattgrid2.dbf'
   SIZE 100M;
   ```

   The output is the following.

   ```
   Tablespace created.
   ```

   See Create the Oracle database users and default tablespace in the *Oracle TimesTen In-Memory Database Cache Guide*.

2. Use SQL*Plus to create the schema user. Grant this schema user the minimum privileges required to create tables in the Oracle Database to be cached in your TimesTen database.

   This example creates the `oratt_grid` schema user.

   ```
   CREATE USER oratt_grid IDENTIFIED BY oraclepwd
   DEFAULT TABLESPACE cachegridtablespace1 QUOTA UNLIMITED ON
   cachegridtablespace1;
   ```

   The output is the following.

   ```
   User created.
   ```

   ```
   GRANT CREATE SESSION, RESOURCE TO oratt_grid;
   ```

   The output is the following.

   ```
   Grant succeeded.
   ```

3. Use SQL*Plus to create the cache administration user. Later, you will use the same name in the `cacheUser` metadata file.

This example creates the `cacheuser_grid` user.

```
CREATE USER cacheuser_grid IDENTIFIED BY oraclepwd
DEFAULT TABLESPACE cachegridtablespace2 QUOTA UNLIMITED ON
cachegridtablespace2;
```

The output is the following.

```
User created.
```

Commit and then exit SQL*Plus.

```
commit;
```

```
exit;
```

Exit the shell.

```
exit;
```

Grant privileges to the cache administration user

The cache administration user needs a specific set of privileges to work with TimesTen cache groups. TimesTen provides the `grantCacheAdminPrivileges.sql` SQL*Plus script as part of the TimesTen installation distribution. You run this script in your Oracle database to grant the cache administration user the minimum set of privileges required to perform cache operations. See Grant privileges to the Oracle database users and see Required privileges for cache operations in the *Oracle TimesTen In-Memory Database Cache Guide*.

Perform these steps to run the `grantCacheAdminPrivileges.sql` script:

1. Create a shell from which you can access your Oracle Database, and then from the directory of your choice, create an empty subdirectory. This example creates the `oraclescripts` subdirectory.

   ```
   mkdir -p oraclescripts
   ```

2. From your Linux development host, use the `kubectl cp` command to copy the `grantCacheAdminPrivileges.sql` script from the `*installation_dir*/oraclescripts` directory on your Linux development host to the `oraclescripts` directory that you just created.

Note:

You must unpack the TimesTen distribution to retrieve the `installation_dir/oraclescripts/grantCacheAdminPrivileges.sql` directory. See Unpack the TimesTen and the TimesTen Kubernetes Operator distributions.

```
kubectl cp /installation_dir/oraclescripts/
grantCacheAdminPrivileges.sql
database-oracle:oraclescripts/grantCacheAdminPrivileges.sql
```

3. (Optional): From your shell, verify the script is located in the `oraclescripts` directory.

```
ls oraclescripts
```

The output is the following.

```
grantCacheAdminPrivileges.sql
```

4. Change to the `database-oracle:oraclescripts` directory. Next, use SQL*Plus to connect to the Oracle Database as the `sys` user. Next, run the `grantCacheAdminPrivileges.sql` script.

```
cd oraclescripts
```

```
sqlplus sys/syspwd@oracache as sysdba
```

In SQL*Plus, run the SQL script.

```
@grantCacheAdminPrivileges "cacheuser_grid";
```

The output is similar to the following.

```
Please enter the administrator user id
The value chosen for administrator user id is cacheuser_grid

TT_CACHE_ADMIN_ROLE role already exists
***************** Initialization for cache admin begins
******************
0. Granting the CREATE SESSION privilege to CACHEUSER_GRID
1. Granting the TT_CACHE_ADMIN_ROLE to CACHEUSER_GRID
2. Granting the DBMS_LOCK package privilege to CACHEUSER_GRID
3. Granting the DBMS_DDL package privilege to CACHEUSER_GRID
4. Granting the DBMS_FLASHBACK package privilege to CACHEUSER_GRID
5. Granting the CREATE SEQUENCE privilege to CACHEUSER_GRID
6. Granting the CREATE CLUSTER privilege to CACHEUSER_GRID
7. Granting the CREATE OPERATOR privilege to CACHEUSER_GRID
8. Granting the CREATE INDEXTYPE privilege to CACHEUSER_GRID
```

```
9. Granting the CREATE TABLE privilege to CACHEUSER_GRID
10. Granting the CREATE PROCEDURE  privilege to CACHEUSER_GRID
11. Granting the CREATE ANY TRIGGER  privilege to CACHEUSER_GRID
12. Granting the GRANT UNLIMITED TABLESPACE privilege to CACHEUSER_GRID
13. Granting the DBMS_LOB package privilege to CACHEUSER_GRID
14. Granting the SELECT on SYS.ALL_OBJECTS privilege to CACHEUSER_GRID
15. Granting the SELECT on SYS.ALL_SYNONYMS privilege to CACHEUSER_GRID
16. Checking if the cache administrator user has permissions on the
default
tablespace
     Permission exists
18. Granting the CREATE TYPE privilege to CACHEUSER_GRID
19. Granting the SELECT on SYS.GV$LOCK privilege to CACHEUSER_GRID
20. Granting the SELECT on SYS.GV$SESSION privilege  to CACHEUSER_GRID
21. Granting the SELECT on SYS.DBA_DATA_FILES privilege  to CACHEUSER_GRID
22. Granting the SELECT on SYS.USER_USERS privilege  to CACHEUSER_GRID
23. Granting the SELECT on SYS.USER_FREE_SPACE privilege  to
CACHEUSER_GRID
24. Granting the SELECT on SYS.USER_TS_QUOTAS privilege  to CACHEUSER_GRID
25. Granting the SELECT on SYS.USER_SYS_PRIVS privilege  to CACHEUSER_GRID
26. Granting the SELECT on SYS.V$DATABASE privilege  to CACHEUSER_GRID
(optional)
27. Granting the SELECT on SYS.GV$PROCESS privilege  to CACHEUSER_GRID
(optional)
28. Granting the SELECT ANY TRANSACTION privilege to CACHEUSER_GRID
29. Creating the CACHEUSER_GRID.TT_07_ARDL_CG_COUNTER table
30. Granting SELECT privilege on CACHEUSER_GRID.TT_07_ARDL_CG_COUNTER
table to
PUBLIC
********* Initialization for cache admin user done successfully *********
```

Exit from SQL*Plus and the shell.

```
exit;
```

```
exit;
```

You successfully ran the script to grant privileges to the cache administration user.

Create the Oracle Database table to be cached

This example creates the `readtab` table in the `oratt_grid` user schema. Later, the table is cached in a read-only cache group.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the `sys` user. Then create the `oratt_grid.readtab` table.

   ```
   sqlplus sys/syspwd@oracache as sysdba
   ```

   ```
   CREATE TABLE oratt_grid.readtab (keyval NUMBER NOT NULL PRIMARY
   KEY,str VARCHAR2(32));
   ```

   The output is the following.

   ```
   Table created.
   ```

2. Use SQL*Plus to insert rows into the `oratt_grid.readtab` table. Next, verify the rows have been inserted.

   ```
   INSERT INTO oratt_grid.readtab VALUES (1,'Hello');
   ```

   ```
   INSERT INTO oratt_grid.readtab VALUES (2,'World');
   ```

   Commit.

   ```
   commit;
   ```

   Verify the rows have been inserted into the tables.

   ```
   SELECT * FROM oratt_grid.readtab;
   ```

   The output is the following.

   ```
   KEYVAL STR
   ---------- --------------------------------
            1 Hello
            2 World
   ```

3. Use SQL*Plus to grant the `SELECT` privilege on the `oratt_grid.readtab` table.

   ```
   GRANT SELECT ON oratt_grid.readtab TO cacheuser_grid;
   ```

   The output is the following.

   ```
   Grant succeeded.
   ```

4. Use SQL*Plus to query the `nls_database_parameters` system view to determine the Oracle Database database character set. The Oracle Database database character set must match the TimesTen database character set.

In this example, the query returns the `AL32UTF8` database character set.

```
 SELECT value FROM nls_database_parameters WHERE
parameter='NLS_CHARACTERSET';
```

The output is the following.

```
VALUE
--------------------------------------------------------------------------
------
AL32UTF8
```

Exit from SQL*Plus and the shell.

```
exit;


exit;
```

You successfully created the Oracle Database table. Later, this table is cached in the TimesTen cache group.

Create the metadata files and the Kubernetes facility

The following metadata files are specific to using TimesTen Cache:

- `cacheUser`: This file is required. The user in this file is created in the TimesTen databases and serves as the cache manager. The name of this user must match the name of the cache administration user that you created in the Oracle Database. See Create the Oracle Database users for information on the cache administration user in the Oracle Database. See cacheUser for more information on the `cacheUser` metadata file.
- `cachegroups.sql`: This file is required. The contents of this file contain the `CREATE CACHE GROUP` definitions. The file can also contain the `LOAD CACHE GROUP` statement and the built-in procedures to update statistics on the cache group tables (such as, `ttOptEstimateStats` and `ttOptUpdateStats`). See cachegroups.sql.
- `tnsnames.ora`: This file is required. It defines Oracle Net Services to which applications connect. For TimesTen Cache, this file configures the connectivity between TimesTen and the Oracle Database (from which data is being cached). In this context, TimesTen is the application that is the connection to the Oracle Database. See tnsnames.ora.
- `sqlnet.ora`: This file may be required. It may be necessary depending on your Oracle Database configuration. The file defines options for how client applications communicate with the Oracle Database. In this context, TimesTen is the application. The `tnsnames.ora` and `sqlnet.ora` files together define how an application communicates with the Oracle Database. See sqlnet.ora.
- `db.ini`: This file is required if you are using TimesTen Cache. The contents of this file contain TimesTen connection attributes for your TimesTen databases, which will be included in TimesTen's `sys.odbc.ini` file. For TimesTen Cache, you must specify the `OracleNetServiceName` and the `DatabaseCharacterSet` connection attributes. The `DatabaseCharacterSet` connection attribute must match the Oracle database character set. See db.ini.

- `schema.sql`: The contents of this file contain database objects, such as tables, sequences, and users. The instance administrator uses the `ttIsql` utility to run this file immediately after the database is created. This file is run before the Operator configures TimesTen Cache or replication, so ensure there are no cache definitions in this file.

  In TimesTen Cache, one or more cache table users own the cache tables. If this cache table user is not the cache manager user, then you must specify the `schema.sql` file and in it you must include the schema user and assign the appropriate privileges to this schema user. For example, if the `oratt_grid` schema user was created in the Oracle Database, and this user is not the TimesTen cache manager user, you must create the TimesTen `oratt_grid` user in this file. See Create the Oracle Database users for more information on the schema users in the Oracle Database. Also see schema.sql for more information on the `schema.sql` file.

In addition, use these metadata files for TimesTenScaleout objects:

- `*.connect`: A file with the `.connect` extension signifies a direct connectable that is used for direct mode access to a database in TimesTen Scaleout. The `*` prefix is the name you choose for the connectable. You can define more than one. See *.connect.
- `*.csconnect`: A file with the `.csconnect` extension signifies a client/server connectable that is used for client/server access to a database in TimesTen Scaleout. The `*` prefix is the name you choose for the connectable. You can define more than one. See *.csconnect.
- `adminUser` (Optional): The user in this file is created in the TimesTen databases and is granted `ADMIN` privileges. See adminUser for more information on this file.

You can include these metadata files in one or more Kubernetes facilities (for example, in a Kubernetes Secret, in a ConfigMap, or in an init container). This ensures the metadata files are populated in the `/ttconfig` directory of the TimesTen containers. Note that there is no requirement as to how to get the metadata files into this `/ttconfig` directory. See Populating the /ttconfig directory for more information.

This example uses the ConfigMap facility to populate the `/ttconfig` directory in your TimesTen containers.

On your Linux development host:

1. From the directory of your choice, create an empty subdirectory for the metadata files. This example creates the `cm_cachegrid` subdirectory.

   ```
   mkdir -p cm_cachegrid
   ```

2. Change to the `cm_cachegrid` directory.

   ```
   cd cm_cachegrid
   ```

3. Create the `adminUser` file in this ConfigMap directory.

   ```
   vi adminUser

   adminuser_grid/adminuserpwd
   ```

4. Create the `db.ini` file in this ConfigMap directory. In this `db.ini` file, define the `PermSize`, `DatabaseCharacterSet`, and the `OracleNetServiceName` connection attributes. The `DatabaseCharacterSet` value must match the database character set value in the Oracle Database. See Create the Oracle Database tables to be cached for information on how to query the `nls_database_parameters` system view to determine the Oracle Database database character set. In this example, the value is `AL32UTF8`.

   ```
   vi db.ini

   permSize=200
   databaseCharacterSet=AL32UTF8
   oracleNetServiceName=Oracache
   ```

5. Create the `schema.sql` file in this ConfigMap directory. In this example, create the `oratt_grid` user. Recall that this user was previously created in the Oracle Database. See Create the Oracle Database users for information on the `oratt_grid` user in the Oracle Database.

   ```
   vi schema.sql

   create user oratt_grid identified by ttpwd;
   grant admin to oratt_grid;
   ```

6. Create the `cacheUser` metadata file in this ConfigMap directory. The `cacheUser` file must contain one line of the form `cacheuser_grid/ttpassword/orapassword`, where `cacheuser_grid` is the user you wish to designate as the cache manager in the TimesTen database, `ttpassword` is the TimesTen password you wish to assign to this user, and `orapassword` is the Oracle Database password that has already been assigned to the Oracle Database cache administration user. Note that the name in this file must match the Oracle Database cache administration user that you previously created. See Create the Oracle Database users for more information on the Oracle Database cache administration user.

   In this example, the `cacheuser_grid` user with password of `oraclepwd` was already created in the Oracle Database. Therefore, supply `cacheuser_grid` as the TimesTen cache manager user. You can assign any TimesTen password to this TimesTen cache manager user. This example assigns `ttpwd`.

   ```
   vi cacheuser_grid

   cacheuser_grid/ttpwd/oraclepwd
   ```

7. Create the `cachegroups.sql` metadata file in this ConfigMap directory. The `cachegroups.sql` file contains the cache group definitions. In this example, a read-only cache group is created. In addition, the `LOAD CACHE GROUP` statement is included to load rows from the `oratt_grid.readtab` table in the Oracle Database into the `oratt_grid.readtab` cache table in the TimesTen database.

   ```
   vi cachegroups.sql

   CREATE READONLY CACHE GROUP readcache
   AUTOREFRESH
     INTERVAL 5 SECONDS
   FROM oratt_grid.readtab (
   ```

```
  keyval NUMBER NOT NULL PRIMARY KEY,
  str VARCHAR2(32)
);

LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
```

8. Create the `tnsnames.ora` metadata file in this ConfigMap directory.

```
vi tnsnames.ora

OraTest =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST =
database.mynamespace.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraTest.my.sample.com)))
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST =
database.mynamespace.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.sample.com)))
```

9. Create the `sqlnet.ora` metadata file in this ConfigMap directory.

```
vi sqlnet.ora


NAME.DIRECTORY_PATH= {TNSNAMES, EZCONNECT, HOSTNAME}
SQLNET.EXPIRE_TIME = 10
SSL_VERSION = 1.2
```

10. Create the direct connectable.

```
vi cachegriddirect.connect

ConnectionCharacterSet=AL32UTF8
```

11. Create the client/server connectable.

```
vi cachegridclient.csconnect

TTC_Timeout=70
```

12. (Optional): Use the Linux `ls` command to verify the metadata files are in the ConfigMap directory.

```
ls
```

The output is the following.

```
adminUser                   cachegroups.sql  schema.sql
cachegridclient.csconnect   cacheUser        sqlnet.ora
cachegriddirect.connect     db.ini           tnsnames.ora
```

13. Create the ConfigMap.

    In this example:

    - The name of the ConfigMap is `cachegrid`. Replace `cachegrid` with a name of your choosing.
    - This example uses `cm_cachegrid` as the directory for the ConfigMap files. If you use a different directory, replace `cm_cachegrid` with the name of your directory.

    ```
    kubectl create configmap cachegrid --from-file .
    ```

    The files in the `cm_cachegrid` directory are included in the ConfigMap and, later, will be available in the TimesTen containers.

14. Use the `kubectl describe` command to verify the contents of the ConfigMap.

    ```
    kubectl describe configmap cachegrid
    ```

    The output is the following.

    ```
    Name:         cachegrid
    Namespace:    mynamespace
    Labels:       <none>
    Annotations:  <none>

    Data
    ====
    cachegroups.sql:
    ----
    CREATE READONLY CACHE GROUP readcache
    AUTOREFRESH
      INTERVAL 5 SECONDS
    FROM oratt_grid.readtab (
      keyval NUMBER NOT NULL PRIMARY KEY,
      str VARCHAR2(32)
    );

    LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;

    db.ini:
    ----
    permSize=200
    databaseCharacterSet=AL32UTF8
    oracleNetServiceName=Oracache

    sqlnet.ora:
    ----
    NAME.DIRECTORY_PATH= {TNSNAMES, EZCONNECT, HOSTNAME}
    ```

```
SQLNET.EXPIRE_TIME = 10
SSL_VERSION = 1.2


tnsnames.ora:
----
OraTest =    (DESCRIPTION =
               (ADDRESS = (PROTOCOL = TCP)(HOST =
database.myhost.svc.cluster.local)
                 (PORT = 1521))
               (CONNECT_DATA =
                 (SERVER = DEDICATED)
                 (SERVICE_NAME = Oratest.my.sample.com)))
OraCache =    (DESCRIPTION =
                (ADDRESS = (PROTOCOL = TCP)(HOST =
database.myhost.svc.cluster.local)
                  (PORT = 1521))
                (CONNECT_DATA =
                  (SERVER = DEDICATED)
                  (SERVICE_NAME = OraCache)))

adminUser:
----
adminuser_grid/adminuserpwd

cachegridclient.csconnect:
----
TTC_Timeout=70

cachegriddirect.connect:
----
ConnectionCharacterSet=AL32UTF8

cacheUser:
----
cacheuser_grid/ttpwd/oraclepwd

schema.sql:
----
create user oratt_grid identified by ttpwd;
grant admin to oratt_grid;

Events:  <none>
```

You successfully created and deployed the `cachegrid` ConfigMap.

Create the TimesTenScaleout object

This section creates the TimesTenScaleout object.

Perform these steps:

1. Create an empty YAML file. You can choose any name, but you may want to use the same name you used for the name of the TimesTenScaleout object. (In this

example, cachegrid.) See TimesTenScaleoutSpecSpec for information on the fields that you must specify in this YAML file as well as the fields that are optional.

Specify these TimesTen Scaleout specific fields:

- k: Set the value of k to the number of copies of data for your TimesTen database. This value determines the number of StatefulSets that the TimesTen Operator creates. A StatefulSet provides the Pods that are used to implement a single data space in the grid. For example, if you set k to 2, the Operator creates two StatefulSets. One StatefulSet provides the Pods for the data instances in data space one. The second StatefulSet provides the Pods for the data instances in data space two.

  This example sets k to 2.

  For information on K-safety and determining an appropriate value for k, see K-safety in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
- nReplicaSets: Set the value to the number of replica sets in the grid. A replica set contains k elements (where each element is an exact copy of the other elements in the replica set). The nReplicaSets value determines the number of replicas for each StatefulSet. For example, if you set k to 2, the TimesTen Operator creates two StatefulSets for the data instances. If you set nReplicaSets to 3, each StatefulSet contains three replicas, and the total number of replica sets in the database is three.

  This example sets nReplicaSets to 3.

  For information on replica sets, see Understanding replica sets in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
- nZookeeper: Set the value to the number of ZooKeeper Pods to provision in a StatefulSet. Your options are 1 or 3 or 5.

  This example sets nZookeeper to 3.

Next, specify these fields:

- name: Replace cachegrid with the name of your TimesTenClassic object.
- storageClassName: Replace oci with the name of the storage class used to allocate PersistentVolumes to hold TimesTen.
- storageSize: Replace 250Gi with the amount of storage that should be requested for each Pod to hold TimesTen. Note: This example assumes a production environment and uses a value of 250Gi for storageSize. For demonstration purposes, a value of 50Gi is adequate.
- image: Replace container-registry.oracle.com/timesten/timesten:22.1.1.9.0 with the location and name of your image.
- imagePullSecret: Replace sekret with the image pull secret that Kubernetes should use to fetch the TimesTen image.
- dbConfigMap: This example uses one ConfigMap (called cachegrid) for the metadata files.

```
vi cachetest.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenScaleout
metadata:
  name: cachegrid
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250Gi
    image: container-registry.oracle.com/timesten/
timesten:22.1.1.9.0
    imagePullSecret: sekret
    dbConfigMap:
    - cachegrid
    k: 2
    nReplicaSets: 3
    nZookeeper: 3
```

2. Create the TimesTenScaleout object.

   ```
   kubectl create -f cachegrid.yaml
   ```

   The output is the following.

   ```
   timestenscaleout.timesten.oracle.com/cachegrid created
   ```

You successfully created the TimesTenScaleout object. The process of deploying your grid and associated databases begins, but is not yet complete.

Monitor the deployment

Use the `kubectl get` and the `kubectl describe` commands to monitor the progress of the grid and its associated databases. There is one database in this example.

1. Use the `kubectl get` command and review the `OVERALL` field. Observe the value is `Initializing`.

   ```
   kubectl get tts cachegrid
   ```

   The output is similar to the following.

   ```
   NAME        OVERALL         MGMT   CREATE   LOAD   OPEN   AGE
   cachegrid   Initializing                                 12s
   ```

   ```
   kubectl get tts cachegrid
   ```

   The output is similar to the following.

   ```
   NAME        OVERALL          MGMT   CREATE   LOAD   OPEN   AGE
   cachegrid   ZookeeperReady                                3m
   ```

   ```
   kubectl get tts cachegrid
   ```

The output is similar to the following.

```
NAME         OVERALL       MGMT    CREATE    LOAD    OPEN    AGE
cachegrid    GridCreated                                     3m54s
```

```
kubectl get tts cachegrid
```

The output is similar to the following.

```
NAME         OVERALL            MGMT    CREATE    LOAD    OPEN    AGE
cachegrid    InstancesCreated                                     6m17s
```

```
kubectl get tts cachegrid
```

The output is similar to the following.

```
NAME         OVERALL           MGMT    CREATE    LOAD    OPEN    AGE
cachegrid    DatabaseCreated                                     6m59s
```

2. Use the `kubectl get` command again to see if the TimesTenScaleout object has transitioned to the `Normal` state. A `Normal` state indicates the grid and database are provisioned, and the process is complete.

   ```
   kubectl get tts cachegrid
   ```

   The output is similar to the following.

   ```
   NAME         OVERALL    MGMT      CREATE     LOAD               OPEN    AGE
   cachegrid    Normal     Normal    created    loaded-complete    open    8m29s
   ```

You successfully deployed your grid and database. Let's verify TimesTen Cache is configured correctly, and is working properly.

Verify TimesTen Cache is configured correctly

To verify that TimesTen Cache is configured correctly and is working properly, perform the following steps:

1. Review the Pods.

   ```
   kubectl get pods
   ```

   The output is similar to the following.

   ```
   NAME                                 READY    STATUS     RESTARTS    AGE
   cachegrid-data-1-0                   2/2      Running    0           33m
   cachegrid-data-1-1                   2/2      Running    0           33m
   cachegrid-data-1-2                   2/2      Running    0           33m
   cachegrid-data-2-0                   2/2      Running    0           33m
   ```

```
cachegrid-data-2-1                    2/2     Running    0
33m
cachegrid-data-2-2                    2/2     Running    0
33m
cachegrid-mgmt-0                      2/2     Running    0
33m
cachegrid-zk-0                        1/1     Running    0
33m
cachegrid-zk-1                        1/1     Running    0
32m
cachegrid-zk-2                        1/1     Running    0
31m
timesten-operator-7677964df9-sp2zp   1/1     Running    0
4d16h
```

2. Use the `kubectl exec -it` command to invoke the shell in one of the data instances (`cachegrid-data-1-0`, in this example).

   ```
   kubectl exec -it cachegrid-data-1-0 -c tt -- /bin/bash
   ```

3. Use `ttIsql` to connect to the `cachegrid` database. Confirm the TimesTen connection attributes are correct. In particular, note that the `OracleNetServiceName` connection attribute is correctly set to `Oracache`.

   ```
   ttIsql cachegrid
   ```

   The output is similar to the following.

   ```
   Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights
   reserved.
   Type ? or "help" for help, type "exit" to quit ttIsql.



   connect "DSN=cachegrid";
   Connection successful: DSN=cachegrid;Durability=0;UID=timesten;
   DataStore=/tt/home/timesten/datastore/
   cachegrid;DatabaseCharacterSet=AL32UTF8;PermSize=200;Connections=100
   ;
   OracleNetServiceName=Oracache;
   (Default setting AutoCommit=1)
   ```

4. Use the `ttIsql cachegroups` to view the cache group.

   ```
   cachegroups;
   ```

   The output is similar to the following.

   ```
   Cache Group CACHEUSER_GRID.READCACHE:

     Cache Group Type: Read Only
     Autorefresh: Yes
     Autorefresh Mode: Incremental
   ```

```
  Autorefresh State: On
  Autorefresh Interval: 5 Seconds
  Autorefresh Status: ok
  Aging: No aging defined

  Root Table: ORATT_GRID.READTAB
  Table Type: Read Only

1 cache group found.
```

5. Use `ttIsql` to query the `oratt_grid.readtab` cache table. Note that the data from the `oratt_grid.readtab` cached table in the Oracle Database is correctly loaded in the `oratt_grid.readcache` cache table in the TimesTen database. Recall that you specified the `LOAD CACHE GROUP` statement in the `cachegroups.sql` metadata file.

   ```
   SELECT * FROM oratt_grid.readtab;
   ```

   The output is similar to the following.

   ```
   < 1, Hello >
   < 2, World >
   2 rows found.
   ```

   Exit `ttIsql` and the shell.

   ```
   exit;
   ```

   ```
   exit;
   ```

You verified that the cache group was created and data was correctly loaded.

Perform operations on the oratt_grid.readtab table

This section performs operations on the `oratt_grid.readtab` table.

1. Create a shell from which you can access your Oracle Database and then use SQL*Plus to connect to the Oracle Database as the schema user (`oratt_grid`, in this example). Then, insert a new row, delete an existing row, and update an existing row in the `oratt_grid.readtab` table of the Oracle Database and commit the changes.

   ```
   sqlplus oratt_grid/oraclepwd@oracache;
   ```

   ```
   INSERT INTO oratt_grid.readtab VALUES (3,'Welcome');
   ```

   ```
   DELETE FROM oratt_grid.readtab WHERE keyval=2;
   ```

   ```
   UPDATE oratt_grid.readtab SET str='Hi' WHERE keyval=1;
   ```

Commit.

```
commit;
```

The output is the following.

```
Commit complete.
```

Since the read-only cache group was created with an autorefresh interval of 5 seconds, the TimesTen `oratt_grid.readtab` cache table in the `readcache` cache group is automatically refreshed after 5 seconds with the committed updates from the cached `oratt_grid.readtab` table of the Oracle Database. The next step is to test that the data was correctly propagated from the Oracle Database to the TimesTen database.

Exit `ttIsql` and exit the shell.

```
exit;
```

```
exit;
```

2. Use the `kubectl exec -it` command to invoke the shell in the container of the Pod that is running a data instance.

```
kubectl exec -it cachegrid-data-1-0 -c tt -- /bin/bash
```

3. Use the TimesTen `ttIsql` utility to connect to the `cachegrid` database. Query the TimesTen `oratt_grid.readtab` table to verify that the table has been updated with the committed updates from the Oracle Database.

```
ttIsql cachegrid
```

The output is similar to the following.

```
Copyright (c) 1996, 2023, Oracle and/or its affiliates. All rights
reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.



connect "DSN=cachegrid";
Connection successful: DSN=cachegrid;Durability=0;UID=timesten;
DataStore=/tt/home/timesten/datastore/cachegrid;
DatabaseCharacterSet=AL32UTF8;PermSize=200;Connections=100;OracleNet
ServiceName=Oracache;
(Default setting AutoCommit=1)


 SELECT * FROM oratt_grid.readtab;
```

The output is the following.

```
< 1, Hi >
< 3, Welcome >
2 rows found.
```

Exit `ttIsql` and exit the shell.

```
exit;
```

```
exit;
```

You have verified that TimesTen Cache is working correctly.

Clean up the cache metadata on the Oracle Database

When you create certain types of cache groups in a TimesTen database, TimesTen stores metadata about that cache group in the Oracle Database. If you later delete that TimesTen database, TimesTen does not automatically delete the metadata in the Oracle Database. As a result, metadata can accumulate on the Oracle Database. See Dropping Oracle Database objects used by cache groups with autorefresh in the *Oracle TimesTen In-Memory Database Cache Guide* for more information.

However, in a Kubernetes environment, if you provide a `cacheUser` metadata file and a `cachegroups.sql` metadata file when you initially create the TimesTenScaleout object, then, by default, the Operator automatically cleans up the Oracle Database metadata if you delete that TimesTenScaleout object.

If you do not want the Operator to automatically clean up the Oracle Database, set the `cacheCleanup` field in the TimesTenScaleout object definition to `false`. See the `cacheCleanup` datum in TimesTenScaleoutSpecSpec.