Oracle® TimesTen In-Memory Database

Kubernetes Operator User's Guide

Release 18.1

F30658-04

May 2021

ORACLE®

Oracle TimesTen In-Memory Database Kubernetes Operator User's Guide, Release 18.1

F30658-04

Contents

Preface ix

Audience ix
Related documents x
Conventions x
Documentation Accessibility xi

What's New xiii

New features in Release 18.1.4.11.0 xiii
New features in Release 18.1.4.4.0 xiv

1 Overview of the Oracle TimesTen Kubernetes Operator

Overview of containers and Kubernetes 1-1
Custom Resource Definition 1-2
Kubernetes Operator 1-2
Introducing the TimesTen Operator 1-2
The TimesTenClassic object type 1-3
Kubernetes objects: named and typed 1-4
The Operator 1-4
Understanding how the Operator functions 1-5
Objects created by the Operator 1-5
StatefulSet 1-6
Service 1-6
Secret 1-6
Pods 1-6
Events 1-6
The TimesTen containers and the TimesTen agent 1-7
Simple deployment of the TimesTen Operator 1-7

2 Setting up the Environment

Prerequisites 2-1
Downloading TimesTen and the TimesTen Operator 2-2
Configuring Kubernetes 2-4
Deploying the TimesTenClassic CRD 2-4
Building the Operator image 2-5
Deploying the Operator 2-6

Customize the Operator 2-6
Verify that the Operator is running 2-7
Building the TimesTen image 2-8

3 Using Configuration Metadata

Understanding the configuration metadata and the Kubernetes facilities 3-1
The supported metadata files 3-1
adminUser file 3-2
cachegroups.sql 3-2
cacheUser 3-3
csWallet 3-4
db.ini file 3-4
epilog.sql 3-5
replicationWallet 3-5
schema.sql file 3-5
sqlnet.ora file 3-5
tnsnames.ora file 3-6
Populating the /ttconfig directory 3-6
Using ConfigMaps and Secrets 3-6
Example using one ConfigMap 3-7
Example using one ConfigMap and one Secret 3-9
Using an init container 3-12
Additional configuration options 3-13
Persistent storage 3-13
Resources specification for the tt and the daemonlog containers 3-14
Pod location 3-15

4 Deploying TimesTen Databases

Understanding the deployment process 4-1
Defining and creating the TimesTenClassic object 4-2
Monitoring the progress of the active standby pair deployment 4-3
Monitor the state of TimesTenClassic 4-3
Verify the underlying objects exist 4-8
Verify connection to the active database 4-8

5 Using TimesTen Databases

Using direct mode applications 5-1
Using Client/Server drivers 5-3

6 Managing and Monitoring Your Active Standby Pairs

Monitoring the health of each pod in the active standby pair 6-1
CatchingUp 6-2
Down 6-2
Healthy 6-2
HealthyActive 6-2
HealthyStandby 6-2

OtherDown ... 6-2
Terminal ... 6-2
Unknown ... 6-2
UpgradeFailed ... 6-3
Monitoring the health of the active standby pair of databases ... 6-3
ActiveDown ... 6-3
ActiveTakeover ... 6-3
BothDown ... 6-4
ConfiguringActive ... 6-4
Failed ... 6-4
Initializing ... 6-4
ManualInterventionRequired ... 6-4
Normal ... 6-4
Reexamine ... 6-4
StandbyCatchup ... 6-5
StandbyDown ... 6-5
StandbyStarting ... 6-5
WaitingForActive ... 6-5
Understanding the BothDown state ... 6-5
Understanding the ManualInterventionRequired state ... 6-7
Bringing up one database ... 6-8
Verify the conditions are met for the database ... 6-9
Set the reexamine value ... 6-11
Suspending the management of a TimesTenClassic object ... 6-15
Overview ... 6-16
Suspend management of the TimesTenClassic object ... 6-16
Locating the Operator ... 6-18
Managing the TimesTen databases ... 6-18
Manually invoke TimesTen utilities ... 6-19
Modify TimesTen connection attributes ... 6-19
Manually edit the db.ini file ... 6-20
Modifying first connection attributes ... 6-22
Modifying general connection attributes ... 6-24
Revert to manual control ... 6-26
Delete an active standby pair of TimesTen databases ... 6-29

7 Working with TimesTen Cache

Overview ... 7-1
Creating the metadata files and the Kubernetes facility ... 7-3
Creating the TimesTenClassic object ... 7-6
Monitoring the deployment of the TimesTenClassic object ... 7-7
Cleaning up the cache metadata on the Oracle Database ... 7-10

8 Using Encryption for Data Transmission

Creating TLS certificates for replication and Client/Server ... 8-1
Configuring TLS for replication ... 8-3

Create the metadata files and the Kubernetes facilities ... 8-3
Create the Kubernetes Secret ... 8-4
Create the ConfigMap ... 8-5
Create the TimesTenClassic object ... 8-6
Monitor the deployment of the TimesTenClassic object ... 8-8
Verify that TLS is being used for replication ... 8-9
Configuring TLS for Client/Server ... 8-11
Configuration on the server ... 8-11
Overview of the metadata files and the Kubernetes facilities ... 8-11
Create the Kubernetes Secret for the csWallet metadata file ... 8-12
Create the ConfigMap for the server-side attributes ... 8-13
Create the TimesTenClassic object ... 8-15
Monitor the deployment of the TimesTenClassic object ... 8-16
Configuration on the client ... 8-17
Copy the client wallet ... 8-18
Configure the client-side attributes ... 8-18

9 Handling Failover and Recovery

Handling failover and recovery ... 9-1
An example illustrating the failover and recovery process ... 9-1

10 Performing Upgrades

Overview of the upgrade process ... 10-1
Upgrading the Operator ... 10-2
Download the new release of the TimesTen Operator ... 10-3
Replace the crd.yaml and the service_account.yaml files ... 10-4
Build the new Operator image ... 10-5
Review the current Operator ... 10-6
Update the timestenclassic-operator Deployment ... 10-7
Upgrading TimesTen ... 10-10
Build the new TimesTen image ... 10-10
Check the upgrade strategy for each TimesTenClassic object ... 10-12
Perform an automated upgrade ... 10-14
Modify the TimesTenClassic object: automated upgrade ... 10-14
Monitor the automated upgrade ... 10-17
Perform a manual upgrade ... 10-20
Modify the TimesTenClassic object: manual upgrade ... 10-21
Upgrade the standby database ... 10-23
Failover ... 10-27
Verify the active standby pair of databases are upgraded ... 10-31

11 The TimesTenClassic Object Type

Overview of the TimesTenClassic object type ... 11-1
The TimesTenClassic object type ... 11-1
TimesTenClassic ... 11-2
TimesTenClassicSpec ... 11-2

TimesTenClassicSpecSpec 11-3
TimesTenClassicStatus 11-8

A Active Standby Pair Example

Set up the environment A-1
Download the TimesTen Operator A-2
Configure Kubernetes A-3
Deploy the TimesTenClassic CRD A-4
Build the Operator image A-4
Deploy the Operator A-5
Build the TimesTen image A-7
Create the ConfigMap object A-9
Create the TimesTenClassic object A-10
Monitor deployment A-12
Verify the existence of the underlying objects A-16
Verify the connection to the active TimesTen database A-17
Recover from failure A-18
Cleanup A-19

B TimesTen Cache Example

Setting up the Oracle Database to cache data B-1
Create the Oracle Database users B-1
Grant privileges to the cache administration user B-3
Create the Oracle Database tables to be cached B-4
Creating the metadata files and the Kubernetes facility B-6
Creating the TimesTenClassic object B-11
Monitoring the deployment of the TimesTenClassic object B-12
Verifying that TimesTen Cache is configured correctly B-15
Performing operations on the cache group tables B-16
Perform operations on the oratt.readtab table B-16
Perform operations on the oratt.writetab table B-18
Cleaning up the cache metadata on the Oracle Database B-19

C Run Containers as Non-Root

Overview C-1
Set up the environment C-1
Download the TimesTen Operator C-2
Configure Kubernetes C-3
Deploy the TimesTenClassic CRD C-4
Build the Operator image C-4
Deploy the Operator C-6
Build the TimesTen image C-7
Create the ConfigMap object C-9
Create the TimesTenClassic object C-11
Monitor deployment C-12

Verify the TimesTen container runs as non-root C-16

Index

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) provides the ability to create and deploy highly available replicated pairs of TimesTen databases to a Kubernetes cluster with minimal effort. In addition, the TimesTen Operator provides the ability to automate failure detection and recovery.

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and 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 (as in previous releases).
- 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 Application-Tier Database Cache, or TimesTen Cache, is an Oracle Database Enterprise Edition option. TimesTen Cache is ideal for caching 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 Cache offers all of the functionality and performance of TimesTen Classic, plus the additional functionality for caching Oracle Database tables.
- 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 at:
https://docs.oracle.com/database/timesten-18.1.

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 = -L`*`timesten_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.
{ }	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.

Convention	Meaning
#	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, 181 or 18.1 represents TimesTen Release 18.1.
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.

What's New

This section summarizes the new features of Oracle TimesTen In-Memory Database Release 18.1 that are documented in this guide. It provides links to more information.

New features in Release 18.1.4.11.0

- There is support for automated upgrades of TimesTen. See Chapter 10, "Performing Upgrades" for more information.
- In most cases, the Operator can recover TimesTen if both databases in an active standby pair fail. See "Understanding the BothDown state" on page 6-5 for information.
- In some cases, when the Operator is unable to automatically fix problems with TimesTen, it puts the associated TimesTenClassic object into the `ManualInterventionRequired` state. The Operator takes no further action on the object. This enables you to manually fix TimesTen. You can later request that the Operator resume management of TimesTen. See:
  - "Understanding the ManualInterventionRequired state" on page 6-7 for an overview.
  - "Bringing up one database" on page 6-8 for an example.
- You can suspend the management of your TimesTenClassic object by the Operator. See "Suspending the management of a TimesTenClassic object" on page 6-15 for details.
- You can run direct mode applications in their own containers inside of the Pods in your TimesTenClassic deployment. See "Using direct mode applications" on page 5-1 for more information.
- There is support for these CRD syntax elements. See "TimesTenClassicSpecSpec" on page 11-3 for more information.
  - `agentGetTimeout`
  - `agentPostTimeout`
  - `agentTCPTimeout`
  - `agentTLSTimeout`
  - `bothDownBehavior`
  - `daemonLogSidecar`
  - `imagePullPolicy`
  - `imageUpgradeStrategy`

  - `logStorageSelector`
  - `reexamine`
  - `repCreateStatement`
  - `repPort`
  - `repReturnServiceAttribute`
  - `repStoreAttribute`
  - `stopManaging`
  - `storageSelector`
  - `upgradeDownPodTimeout`
  - `waitingForActiveTimeout`
- There is support for the `awtBehindMb` field in TimesTenClassicStatus of the TimesTenClassic object type. See "TimesTenClassicStatus" on page 11-8 for information.
- It is possible to use the Operator in an environment where nothing is allowed to run as root. See Appendix C, "Run Containers as Non-Root" for information.

New features in Release 18.1.4.4.0

- You can configure and use TimesTen Cache. See Chapter 7, "Working with TimesTen Cache" for details. In addition, there is a complete example of using TimesTen Cache in your Kubernetes environment. See Appendix B, "TimesTen Cache Example" for information.
- You can configure Transport Layer Security (TLS) for replication and for Client/Server. See Chapter 8, "Using Encryption for Data Transmission" for more information.
- You can upgrade the Operator and the TimesTen full distribution to a new release. See Chapter 10, "Performing Upgrades" for details.
- There is support for these metadata files:
  - cachegroups.sql
  - cacheUser
  - csWallet
  - epilog.sql
  - replicationWallet
- You can modify TimesTen connection attributes after you create your TimesTenClassic object. See "Modify TimesTen connection attributes" on page 6-19 for details.
- If you are using TimesTen Cache, you can specify whether the metadata in the Oracle Database should be cleaned up when the TimesTenClassic object is deleted. See the `cacheCleanup` entry in the Table , "TimesTenClassicSpecSpec" and see "Cleaning up the cache metadata on the Oracle Database" on page 7-10 for information.
- You can specify resources requirements for the `tt` and the `daemonlog` containers. See "Resources specification for the tt and the daemonlog containers" on page 3-14 for details.

- There is support for the `pollingInterval` and the `unreachableTimeout` CRD syntax elements. See the `pollingInterval` and the `unreachableTimeout` entries in Table 11–3, " TimesTenClassicSpecSpec" for information.
- The Operator keeps tracks of the individual health of each Pod in the TimesTenClassic active standby pair object. See "Monitoring the health of each pod in the active standby pair" on page 6-1 for details.

1

Overview of the Oracle TimesTen Kubernetes Operator

This chapter provides an overview of containers and Kubernetes. It also discusses the TimesTen Operator.

- Overview of containers and Kubernetes
- Introducing the TimesTen Operator
- Understanding how the Operator functions
- Simple deployment of the TimesTen Operator

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

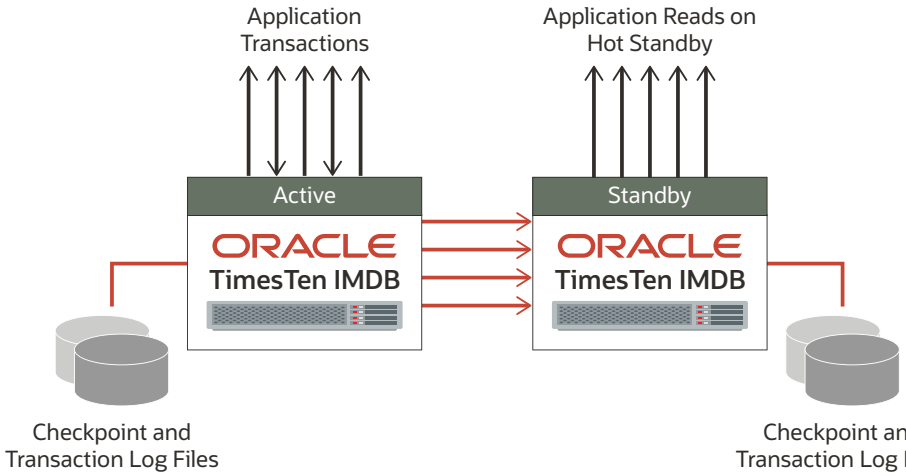
A *Kubernetes Operator* (also called *Operator*) is the brains behind a CRD. An Operator is an application that performs the functions of a human computer operator. It starts, stops, monitors, and manages other applications.

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.

Introducing the TimesTen Operator

TimesTen Classic databases almost always run in active standby pairs. Figure 1–1, "Active standby pair of TimesTen databases" 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.

These sections describe the TimesTenClassic object type and the Operator:

- The TimesTenClassic object type
- The Operator

The TimesTenClassic object type

The TimesTen Operator provides an implementation of the TimesTenClassic CRD, which you install in your Kubernetes cluster. After installation, Kubernetes understands the TimesTenClassic object type, just as it understands Pods, Secrets, and Services.

To create an active standby pair of TimesTen databases in your cluster, you use the `kubectl create` command to create an object of type TimesTenClassic. You define the desired attributes of your TimesTen configuration and your TimesTen database as attributes of this TimesTenClassic object.

Kubernetes objects: named and typed

Objects in Kubernetes are named and typed, so you can define a TimesTenClassic object named `sample` and another TimesTenClassic object named `sample2`. You can have many such Kubernetes objects in a cluster, limited only by the available resources in a Kubernetes cluster.

Objects of different types have different meanings. There can be an object of type `a` called `x` and an object of type `b` called `x` simultaneously. For example, you can define an object of type TimesTenClassic called `sample` and an object of type ConfigMap called `sample` simultaneously. There is no relationship between these two 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.

> **Note:** CRDs are cluster-scoped, not namespace-scoped. There can be different Operators in each namespace, but there can be only one CRD definition for the entire cluster.

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

The Operator

The Operator automatically provisions and configures Pods, configures TimesTen in them, and creates and configures a pair of databases. The Operator monitors the Pods, the TimesTen instances, and the TimesTen databases and keeps them running. For example, in an active standby pair configuration, if the Pod containing the active database fails, the database in the standby Pod is automatically promoted to be the active by the Operator.

This Operator is configured through a Deployment. The `replicas` attribute of the Deployment specifies the number of `replicas` of the Operator that is desired. When you create the Deployment, it causes Kubernetes to create one or more Pods (depending on the number of `replicas`), each of which runs the Operator.

- If you specify `replicas: 1` in the Operator deployment, and the Operator fails, Kubernetes automatically spawns another Operator. When that new Operator starts up, it continues to manage the TimesTenClassic objects within the Deployment's namespace.
- If you specify more than one replica in the Operator deployment, multiple Pods run the Operator. One of these is the active Operator and manages the TimesTenClassic objects in the namespace. The remaining Pods monitor the health of the active Operator. If this active Operator fails, one of the other replicas becomes the active and manages the TimesTenClassic objects within the Deployment's namespace.

Understanding how the Operator functions

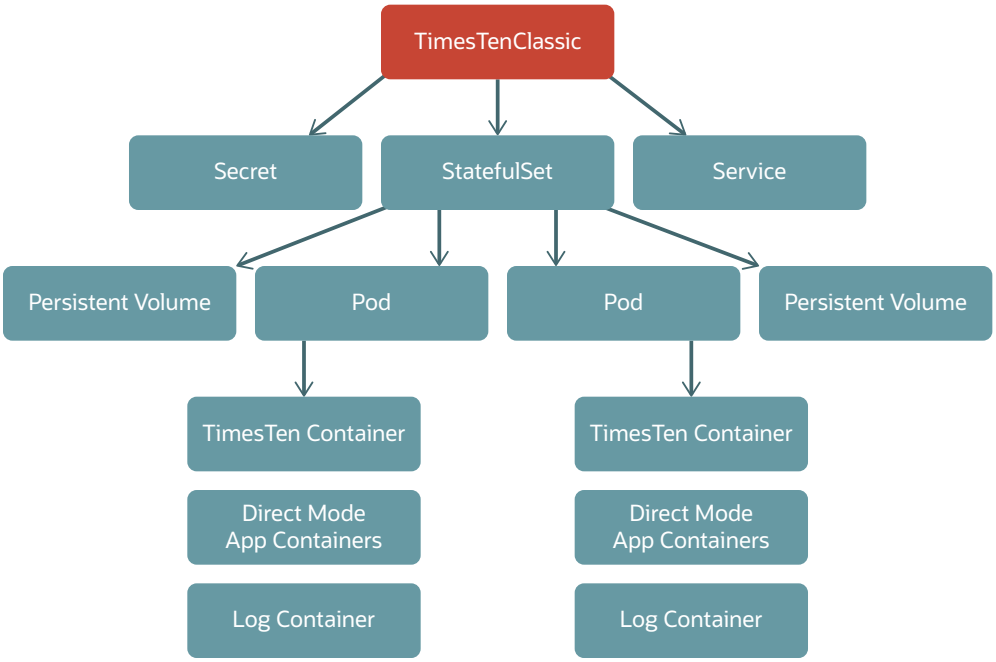
When you create a TimesTenClassic object in the Kubernetes cluster, the process to create and configure your active standby pair of databases begins. The Operator is invoked and creates several Kubernetes objects that are required to run TimesTen. After the objects are created and linked together, the 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:

- Objects created by the Operator
- The TimesTen containers and the TimesTen agent

Objects created by the 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, "Creating the TimesTenClassic object" 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 Operator creates a Secret to inject an SSL certificate into the TimesTen containers. This secures the communication between the Operator and the TimesTen agent.

Pods

The Stateful set 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.

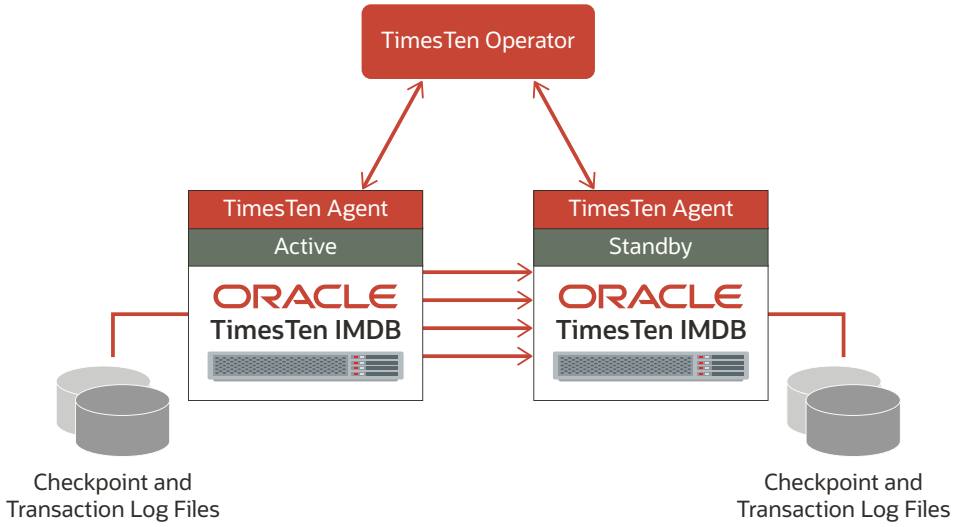
Events

The Operator creates a Kubernetes event whenever important changes occur.

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, "The Operator and the TimesTen agent" 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 of the TimesTen Operator

The TimesTen Operator is designed for simple deployment and 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.

2

Setting up the Environment

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

Note that it is possible to use the Operator in an environment where nothing is allowed to run as root. If this is your desired environment, see Appendix C, "Run Containers as Non-Root" for the procedures to set up such an environment. You do not need to perform the procedures in this chapter if you choose to do the procedures in the Appendix.

Topics:

- Prerequisites
- Downloading TimesTen and the TimesTen Operator
- Configuring Kubernetes
- Deploying the TimesTenClassic CRD
- Building the Operator image
- Deploying the Operator
- Building the TimesTen image

Prerequisites

Complete these prerequisites before installing the TimesTen Operator:

- Ensure you have a working Kubernetes cluster.
  - The Operator and the CRD are developed using the Oracle Cloud Infrastructure Container Engine for Kubernetes (referred to as OKE) with clusters provisioned using *Quick Create*. (OKE release 1.14 or later). See "Introducing the TimesTen Operator" on page 1-2 for information on the Operator and the CRD.)
  - Your cluster must provide a *StorageClass* that can be used to request PersistentVolumes. Each Pod that runs TimesTen uses a PersistentVolume to store the TimesTen database that it manages. 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 other means.
- Ensure you have a Linux 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 to create images and to push the images to the image registry.

- Ensure you have access to an image registry.
  - You need an image registry to run containers in a Kubernetes cluster. You use this registry to store container images that will then be run by Kubernetes. For example, when you use a Kubernetes cluster in OKE, you may consider using the Oracle Container Image Registry (OCIR). You can use other image registries, as well.
  - The development host needs to be able to push images to the registry using the `docker push` command.
  - The Kubernetes cluster needs to be able to pull images from the registry using an *image pull secret*.
  - You must be able to pull base operating system images from image registries.

Downloading TimesTen and the TimesTen Operator

You must download the TimesTen full distribution on Linux-64 bit in order to use the TimesTen Operator.

Perform these steps to download the full distribution of TimesTen and then unpack the TimesTen Operator distribution that is embedded within it. Perform all steps from your Linux development host.

1. From the directory of your choice:
   - Create one subdirectory into which you will download the TimesTen full distribution. For example, create the *`installation_dir`* subdirectory. (The *`installation_dir`* directory is used in the remainder of this book.)
   - Create a second subdirectory into which you will unpack the TimesTen Operator distribution. For example, create the *`kube_files`* subdirectory. (This *`kube_files`* directory is used in the remainder of this book.)

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

   You are now ready to download and unpack the TimesTen full distribution.

2. Navigate to *`installation_dir`*.

   ```
   % cd installation_dir
   ```

   Download the TimesTen full distribution into this directory. As an example, download the `timesten1814110.server.linux8664.zip` file, (the `18.1.4.11.0` full distribution for Linux 64-bit).

3. From the *`installation_dir`*, use the ZIP utility to unpack the TimesTen distribution.

   ```
   % unzip timesten1814110.server.linux8664.zip
   Archive:  /timesten/installation/timesten1814110.server.linux8664.zip
      creating: tt18.1.4.11.0/
      creating: tt18.1.4.11.0/ttoracle_home/
   ```

```
...
   creating: tt18.1.4.11.0/kubernetes/
...
```

You successfully unpacked the TimesTen full distribution.

Note that the *installation_dir*/tt18.1.4.11.0/kubernetes directory is created. The operator.zip file is located in this directory. For example, this is a sample directory structure after unpacking the distribution:

```
% pwd
installation_dir/tt18.1.4.11.0
% dir
3rdparty  include     lib      oraclescripts  README.html  ttoracle_home
bin       info        network  PERL           startup
grid      kubernetes  nls      plsql          support
```

4. Navigate to the *kube_files* directory and unpack the operator.zip file into it. In this example, unpack the *installation_dir*/tt18.1.4.11.0/kubernetes/operator.zip file.

```
% cd kube_files
% unzip installation_dir/tt18.1.4.11.0/kubernetes/operator.zip
[...UNZIP OUTPUT...]
```

You successfully unpacked the *installation_dir*/tt18.1.4.11.0/kubernetes/operator.zip file into the *kube_files* directory.

5. Review the directory structure. Later in this chapter, you will modify some of the files in these subdirectories. This example shows the most important subdirectories and files, which can change from release to release.

```
README.md
deploy/crd.yaml
deploy/operator.yaml
deploy/service_account.yaml
operator/Dockerfile
operator/timestenclassic-operator
ttimage/agent2
ttimage/.bashrc
ttimage/create1.sql
ttimage/create2.sql
ttimage/Dockerfile
ttimage/get1.sql
ttimage/pausecq.sql
ttimage/repcreate.sql
ttimage/repduplicate.sql
ttimage/runsql,sql
ttimage/starthost.pl
ttimage/.ttdotversion
ttimage/.ttdrop
```

Note: This directory tree must persist through the lifetime of the TimesTen Operator.

In addition, do not delete the TimesTen full distribution file (`timesten1814110.server.linux8664.zip`, in this example). You need to copy this file into the:

- `/operator` directory to build the Operator image and push the image to the image registry. See "Building the Operator image" on page 2-5 for details.
- `/ttimage` directory to build the TimesTen image and push the image to the image registry. See "Building the TimesTen image" on page 2-8 for details.

You successfully downloaded and unpacked the TimesTen Operator distribution.

Configuring Kubernetes

The Operator runs by using a Kubernetes *service account*. This service account needs permissions and privileges in your namespace. These permissions and privileges are granted through a *role*. The `service_account.yaml` file adds the service account and the role to your namespace, and grants the service account the privileges that are specified in the role. The `service_account.yaml` file is provided in the `operator.zip` file you previously unpacked.

Perform these steps:

1. Navigate to the *`kube_files`*`/deploy` directory.

   ```
   % cd kube_files/deploy
   ```

2. Create the service account.

   ```
   % kubectl create -f service_account.yaml
   role.rbac.authorization.k8s.io/timestenclassic-operator created
   serviceaccount/timestenclassic-operator created
   rolebinding.rbac.authorization.k8s.io/timestenclassic-operator created
   ```

The `service_account.yaml` file created the `timestenclassic-operator` service account and the `timestenclassic-operator` role in your namespace, and granted the service account the privileges specified in the role.

Deploying the TimesTenClassic CRD

Kubernetes supports objects, such as Pods and StatefulSets. The Kubernetes API can be extended to create customized object types. This step adds a new object type, called TimesTenClassic, to your cluster.

Navigate to the *`kube_files`*`/deploy` directory, and then use the `kubectl create` command to create the TimesTenClassic customized resource definition (CRD) in your Kubernetes cluster.

```
% cd kube_files/deploy
% kubectl create -f crd.yaml
customresourcedefinition.apiextensions.k8s.io/
timestenclassics.timesten.oracle.com created
```

You successfully added the TimesTenClassic object type to your Kubernetes cluster.

Building the Operator image

Kubernetes Operators are Pods that run a customized image. Before you can run the Operator, you must build this image and push it to your image registry.

The files needed to create the image are provided in the *kube_files*/operator directory (part of the ZIP file you previously unpacked). In the *kube_files*/operator directory are the Dockerfile and the binaries needed to create the Operator image.

To build the Operator image and push it to your registry, perform these steps:

1. Navigate to the *kube_files*/operator directory, and copy the TimesTen distribution into it. This example assumes you downloaded the timesten1814110.server.linux8664.zip distribution into the *installation_dir* directory. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for information. Then, verify the timesten1814110.server.linux8664.zip file is in the *kube_files*/operator directory.

```
% cd kube_files/operator
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls -a
Dockerfile
timesten1814110.server.linux8664.zip
timestenclassic-operator
```

2. Navigate to the *kube_files*/operator directory (if not already in this directory) and use the docker command to build the Operator image. You can choose any name for ttclassic-operator:3 (represented in **bold** in this example). Note that the output may change from release to release.

```
% cd kube_files/operator
% docker build -t ttclassic-operator:3 .
Sending build context to Docker daemon  478.6MB
Step 1/7 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/7 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/7 : RUN yum -y install openssl unzip && /usr/sbin/useradd -d
/tt-operator -m -u 1001 -s /bin/nologin -U tt-operator
 ---> Using cache
 ---> e3f1427246ab
Step 4/7 : COPY --chown=tt-operator:tt-operator timestenclassic-operator
/usr/local/bin/timestenclassic-operator
 ---> Using cache
 ---> 6ccad53230f0
Step 5/7 : COPY --chown=tt-operator:tt-operator $TT_DISTRO /tt-operator/
$TT_DISTRO
 ---> 5cd31705485a
Step 6/7 : USER tt-operator
 ---> Running in 6a773ddac5dd
Removing intermediate container 6a773ddac5dd
 ---> 875ee38ebc75
Step 7/7 : ENTRYPOINT ["/usr/local/bin/timestenclassic-operator"]
 ---> Running in fed0f6c94c2f
Removing intermediate container fed0f6c94c2f
 ---> 10dde79e1617
Successfully built 10dde79e1617
```

```
Successfully tagged ttclassic-operator:3
```

3. Use the `docker` command to tag the Operator image.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous step. (`ttclassic-operator:3` is represented in **bold** in this example.)

```
% docker tag ttclassic-operator:3 phx.ocir.io/youraccount/ttclassic-operator:3
```

4. Use the `docker` command to push the Operator image to your registry.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous steps. (`ttclassic-operator:3` is represented in **bold** in this example.)

```
% docker push phx.ocir.io/youraccount/ttclassic-operator:3
The push refers to repository [phx.ocir.io/youraccount/ttclassic-operator]
46458e9fc890: Pushed
471a399f0540: Pushed
9e51a2b82af3: Pushed
2f915858a916: Layer already exists
3: digest:
sha256:9b941f12e3d52298b9b38f7766ddcdfb1d011857a990ff01a8adafd32f3d3e8d size:
1166
```

You successfully built the Operator image and pushed it to your image registry.

Deploying the Operator

This section covers the steps to customize, and then deploy the Operator. It also provides the commands to verify that the Operator is running.

- Customize the Operator
- Verify that the Operator is running

Customize the Operator

To customize the Operator for your namespace, navigate to the *kube_files*/deploy directory, and edit the `operator.yaml` file. This file is provided in the distribution that you previously unpacked. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for details.

1. Modify these fields represented in **bold** (in the `operator.yaml` file below):
   - `replicas: 1`

     Replace `1` with the number of copies of the Operator that you would like to run. `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 Kubernetes uses to pull images from your registry.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry.
   - Replace `ttclassic-operator:3` with the name you chose in the previous steps.

```
% cd kube_files/deploy
% vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timestenclassic-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timestenclassic-operator
  template:
    metadata:
      labels:
        name: timestenclassic-operator
    spec:
      serviceAccountName: timestenclassic-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timestenclassic-operator
          image: phx.ocir.io/youraccount/ttclassic-operator:3
          command:
          - timestenclassic-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "timestenclassic-operator"
            - name: GODEBUG
              value: "x509ignoreCN=0"
```

2. Use the `kubectl create` command to define the Operator to your namespace and to start the Operator.

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

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

Verify that the Operator is running

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

```
% kubectl get pods
NAME                                       READY   STATUS    RESTARTS   AGE
timestenclassic-operator-846cb5c97c-76zbx  1/1     Running   0          32s
```

Building the TimesTen image

Before you can start TimesTen in your Kubernetes cluster, you must first package TimesTen as a container image and then push the image to your image registry. The files that you need to do this are provided in the *kube_files* directory tree. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for information.

To build the TimesTen container image, perform these steps:

1. Navigate to the *kube_files*/ttimage directory, and copy the TimesTen distribution into it. This example assumes you downloaded the timesten1814110.server.linux8664.zip distribution into the *installation_dir* directory. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for information. Then, verify the timesten1814110.server.linux8664.zip file is in the *kube_files*/ttimage directory.

```
% cd kube_files/ttimage
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls *.zip
timesten1814110.server.linux8664.zip
```

2. Navigate to the *kube_files*/ttimage directory (if not already in this directory). Edit the Dockerfile, replacing timesten1814110.server.linux8664.zip with the name of your TimesTen full distribution. If your TimesTen distribution is timesten1814110.server.linux8664.zip, no modification is necessary. If not, the modification you need to make is represented in **bold**. Note: The TimesTen full distribution must be 18.1.4.11.0 or later.

```
% cd kube_files/ttimage
% vi Dockerfile

# Copyright (c) 2019, 2021, Oracle and/or its affiliates.

FROM container-registry.oracle.com/os/oraclelinux:7

ARG TT_DISTRO=timesten1814110.server.linux8664.zip

RUN yum -y install tar gzip vim curl unzip libaio util-linux
RUN groupadd -g 333 oracle
RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle oracle
RUN install -d -m 0750 -o oracle -g oracle /home/oracle
COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop .ttdotversion agent2
create1.sql create2.sql get1.sql repcreate.sql repduplicate.sql runsql.sql
pausecg.sql /home/oracle/
# Uncomment the following line if you are using the optional non-root
installation procedure.
# USER 333
ENTRYPOINT "/home/oracle/starthost.pl"
```

3. Use the docker command to build the TimesTen container image. Replace tt1814110:3 with a name of your choosing (represented in **bold**, in the docker build command below). Note that the output may change from release to release.

```
% docker build -t tt1814110:3 .

Sending build context to Docker daemon  445.8MB
Step 1/9 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/9 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
```

```
 ---> a259a93fe906
Step 3/9 : RUN yum -y install tar gzip vim curl unzip libaio util-linux
 ---> Using cache
 ---> ac676b5376f3
Step 4/9 : RUN groupadd -g 333 oracle
 ---> Using cache
 ---> ce16920f085c
Step 5/9 : RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle
oracle
 ---> Using cache
 ---> 0319814aca1c
Step 6/9 : RUN install -d -m 0750 -o oracle -g oracle /home/oracle
 ---> Using cache
 ---> c8612b53398a
Step 7/9 : COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
 ---> 31cae98b71fd
Step 8/9 : COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop
.ttdotversion agent2 create1.sql create2.sql get1.sql repcreate.sql
repduplicate.sql runsql.sql pausecg.sql /home/oracle/
 ---> e50eb99c9b54
Step 9/9 : ENTRYPOINT "/home/oracle/starthost.pl"
 ---> Running in 0b41efd38837
Removing intermediate container 0b41efd38837
 ---> 171245e546d5
Successfully built 171245e546d5
Successfully tagged tt1814110:3
```

4. Use the `docker` command to tag the TimesTen container image. Replace the following, represented in **bold**, in the `docker tag` command below.
   - `tt1814110:3` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

   ```
   % docker tag tt1814110:3 phx.ocir.io/youraccount/tt1814110:3
   ```

5. Use the `docker` command to push the TimesTen container image to your registry. Replace the following, represented in **bold**, in the `docker push` command below.
   - `phx.ocir.io/youraccount` with the location of your image registry.
   - `tt1814110:3` with the name you chose previously.

   ```
   % docker push phx.ocir.io/youraccount/tt1814110:3

   The push refers to repository [phx.ocir.io/youraccount/tt1814110]
   97a0f250b2fe: Pushed
   650b003a3ad4: Pushed
   b8de51528854: Pushed
   62192d26e325: Pushed
   7dfe13e9b5a4: Pushed
   d8570fce965c: Pushed
   2f915858a916: Layer already exists
   3: digest:
   sha256:a6ac313394229eb2256d4a56fbcd8e2eda50ea2cc21991fa76f11701f2299710
   size: 1788
   ```

You successfully built the TimesTen container image. It is pushed to your image registry.

3

Using Configuration Metadata

This chapter discusses the configuration metadata that you provide to define the attributes of your TimesTen database. This configuration metadata is used by TimesTen and the Operator when TimesTen runs in the Kubernetes cluster. The chapter also discusses the Kubernetes facilities that you can use to get the configuration metadata into your TimesTen containers. The chapter also includes various examples that show you how to define the configuration metadata, and how to use the Kubernetes facilities. It then discussed additional configuration options.

- Understanding the configuration metadata and the Kubernetes facilities
- The supported metadata files
- Populating the /ttconfig directory
- Additional configuration options

Understanding the configuration metadata and the Kubernetes facilities

Configuration metadata, in the form of metadata files, enables you to specify the attributes of your TimesTen database, and how that database is to interact with other applications and components. Each metadata file has a specific name. You create this file and add specific metadata to it. For example, the TimesTen Operator provides support for a file named `db.ini` for the TimesTen connection attributes. You create this file, and in it you include your database's specific connection attributes. See "The supported metadata files" on page 3-1 for details.

Kubernetes supports various facilities to enable you to get the metadata files into the TimesTen containers. Specifically, when the Operator creates each Pod, that Pod has a container that runs TimesTen. This container accesses the metadata files by looking for their existence in the `/ttconfig` directory. By using a Kubernetes facility, the metadata files are placed in the `/ttconfig` directory of the TimesTen containers. See "Populating the /ttconfig directory" on page 3-6 for information on these facilities.

The supported metadata files

These are the supported metadata files. Use these files to specify the attributes and the metadata for your database. After you create these files, and you choose a facility to get these files in your TimesTen containers, TimesTen can then access them to determine the attributes and the metadata that is specific to your database.

These metadata files apply to all databases:

- adminUser file
- db.ini file

- schema.sql file

These metadata files are specific to TimesTen Cache:

- cachegroups.sql
- cacheUser
- sqlnet.ora file
- tnsnames.ora file

These metadata files are specific to TLS support:

- csWallet
- replicationWallet

The epilog.sql metadata file is used for operations that occur after the replication scheme has been created.

adminUser file

The Operator can automatically create a named user with `ADMIN` privileges in your database when it is created. Create the `adminUser` file for this purpose. This file should contain one line of the form:

```
user/password
```

cachegroups.sql

If you are using TimesTen Cache, you must specify the `cachegroups.sql` file. This file contains the create cache group and the load cache group definitions. Specifically, in this file, you specify `CREATE CACHE GROUP` statements. In addition, if you want to load data from the Oracle Database into your cache groups, you can specify one or more `LOAD CACHE GROUP` statements. You can also specify the `ttOptUpdateStats` or the `ttOptEstimateStats` TimesTen built-in procedures to update statistics on the cache tables after the `LOAD CACHE GROUP` operation completes. Ensure these built-in procedures follow the `LOAD CACHE GROUP` statements in the `cachegroups.sql` file.

This file is required as cache groups must be created before replication can be configured.

For more information, see:

- "CREATE CACHE GROUP" and "LOAD CACHE GROUP" in the *Oracle TimesTen In-Memory Database SQL Reference*.
- "ttOptUpdateStats" and "ttOptEstimateStats" in the *Oracle TimesTen In-Memory Database Reference*.
- "Cache group types" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

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

If you are using TimesTen Cache, you must specify the `cacheUser` metadata file. This file must contain one line of the form:

```
cacheUser/ttPassword/oraPassword
```

The `cacheUser` is the user you want to designate as the TimesTen cache manager user. This user must have the same name as the user whom you designated as the cache administration user in the Oracle Database. This user must already exist in the Oracle database. Specify `ttPassword` as the TimesTen password for the TimesTen `cacheUser` user (the TimesTen cache manager). The `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`. Also assume you want to 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
```

In this example, the 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. (Note that you do not need to create this TimesTen user. The Operator does it for you.) See "Create the TimesTen users" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for information on the TimesTen users. Also see "Overview" on page 7-1 and "Creating the metadata files and the Kubernetes facility" on page 7-3 in this book.

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

By default, in a TimesTen Client/Server environment, data is transmitted between your client applications and your TimesTen database unencrypted. 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/oracle/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" on page 8-1 and "Configuring TLS for Client/Server" on page 8-11 for details.

The client wallet must also be available to your client applications. See "Creating TLS certificates for replication and Client/Server" on page 8-1 and "Configuring TLS for Client/Server" on page 8-11 for details.

db.ini file

The `db.ini` file contains the TimesTen connection attributes for your TimesTen database. The connection attributes you specify in the `db.ini` file will be included in TimesTen's `sys.odbc.ini` file. You can specify data store attributes, first connection attributes, and general connection attributes in the `db.ini` file, except do not specify the `DataStore` or the `LogDir` connection attributes. These two attributes are set by the Operator. 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`.)

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

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
```

If you are using TimesTen Cache, ensure you specify the `db.ini` file.

This example shows a sample `db.ini` file, which contains the connection attributes for your TimesTen database:

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

epilog.sql

Use this file for 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 call the `ttXlaBookmarkCreate` built-in procedure in this file.

Here is an example of an `epilog.sql` file. The example calls the `ttXlaBookmarkCreate` built-in procedure to create XLA bookmarks. See the "ttXlaBookmarkCreate" built-in procedure in the *Oracle TimesTen In-Memory Database Reference* for more information.

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

replicationWallet

By default, TimesTen replication transmits data between your TimesTen databases unencrypted. However, you can configure TLS for replication to ensure secure network communication between your replicated TimesTen databases. To do this, you must 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" on page 8-1 and "Configuring TLS for replication" on page 8-3 for details.

You must also include the `replicationCipherSuite` field and optionally include the `replicationSSLMandatory` field in your TimesTenClassic object definition. See the `replicationCipherSuite` entry and the `replicationSSLMandatory` entry in Table 11–3, " TimesTenClassicSpecSpec" and see "Configuring TLS for replication" on page 8-3 for details.

schema.sql file

The 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 instance administrator runs this file (by using the `ttIsql` utility) immediately after the database is created. The file is run 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" on page B-1 for more information on the schema users in the Oracle Database. Also see "Create the TimesTen users" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

Do not include cache definitions in this file. Instead, use the `cachegroups.sql` metadata file. See "cachegroups.sql" on page 3-2 for information.

sqlnet.ora file

The Oracle Database `sqlnet.ora` file defines 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 file" on page 3-6 for information on the `tnsnames.ora` file.

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 file

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 file) if you are using:

- TimesTen Cache
- SQL APIs, such as Pro*C, OCI, or ODPI-C
- The `ttLoadFromOracle` feature (See "ttLoadFromOracle" in the *Oracle TimesTen In-Memory Database Reference* for more information).

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

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

Populating the /ttconfig directory

To configure TimesTen with the supported metadata files, you must ensure the files are placed in the `/ttconfig` directory of the TimesTen containers. (See "The supported metadata files" on page 3-1 for details on the supported metadata files.)

There is no requirement as to how you get the files into the `/ttconfig` directory. However, Kubernetes does provide these facilities for you to consider:

- Using ConfigMaps and Secrets
- Using an init container

Using ConfigMaps and Secrets

You can use one or more ConfigMaps and (or) one or more Secrets to incorporate the metadata files into the TimesTen containers. This enables you to give different deployments of TimesTen different metadata by using different objects for each deployment. 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.

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, as another 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 object definition. See "Understanding the deployment process" on page 4-1 for a detailed explanation of how to create your TimesTenClassic object. In the following sections, there are examples that illustrate how and where to reference the ConfigMaps and Secrets in your TimesTenClassic object definition (in your YAML file). But, see "Understanding the deployment process" on page 4-1 for the details of how to create the TimesTenClassic object.

  When you use ConfigMaps and Secrets to hold your metadata and then reference them in the TimesTenClassic object definition, the Operator creates a *ProjectedVolume* 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 object. This volume is mounted as `/ttconfig` in the TimesTen containers.

Here are two examples illustrating how to use ConfigMaps and Secrets:

- Example using one ConfigMap
- Example using one ConfigMap and one Secret

Example using one ConfigMap

This example uses one ConfigMap (called `sample`) for 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_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 `scott` user with the `tiger` password.

```
vi adminUser

scott/tiger
```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

```
vi schema.sql

create sequence scott.s;
create table scott.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
```

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:
----
scott/tiger

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

schema.sql:
----
create sequence scott.s;
```

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


Events:  <none>
```

You successfully created and deployed the `sample` ConfigMap.

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

   Note this example uses a `storageSize` of `250G` (suitable for a production environment). For demonstration purposes, a `storageSize` of `50G` is adequate. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youracount/tt1814110:3
    imagePullSecret: sekret
    dbConfigMap:
    - sample
```

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

   See "TimesTenClassicSpecSpec" on page 11-3 for information on the `dbConfigMap` attribute. See "Defining and creating the TimesTenClassic object" on page 4-2 for information on creating the TimesTenClassic object.

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.

On your Linux development host:

1. 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. Navigate 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. Navigate 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 `scott` user with the `tiger` password.

```
vi adminUser

scott/tiger
```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

```
vi schema.sql

create sequence scott.s;
create table scott.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. (`myconfig` is represented in **bold** in this example.)
   - 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. (`cm_myconfig` is represented in **bold** in this example.)

   Use the `kubectl create` command to create the ConfigMap:

```
% kubectl create configmap myconfig --from-file=cm_myconfig
configmap/myconfig created
```

   You successfully created and deployed the `myconfig` ConfigMap.

8. Use the `kubectl describe` command to verify the contents of the ConfigMap. (`myconfig`, in this example.)

```
% kubectl describe configmap myconf
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. (`mysecret` is represented in **bold** in this example.)
   - 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. (`secret_mysecret` is represented in **bold** in this example.)

   Use the `kubectl create` command to create the Secret:

   ```
   % kubectl create secret generic mysecret --from-file=secret_mysecret
   secret/mysecret created
   ```

   You successfully created and deployed the `mysecret` Secret.

10. Use the `kubectl describe` command to view the Secret. (`mysecret`, in this example.) Note the contents of the `adminUser` and the `schema.sql` files are not displayed.

    ```
    % kubectl describe secret mysecret
    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 TimesTenClassic object definition.

    - In the `dbConfigMap` field, specify the name of the your ConfigMap (`myconfig`, in this example, represented in **bold**).
    - In the `dbSecret` field, specify the name of the your Secret (`mysecret`, in this example, represented in **bold**).

    ```
    apiVersion: timesten.oracle.com/v1
    kind: TimesTenClassic
    metadata:
      name: sample
    spec:
      ttspec:
        storageClassName: oci
        storageSize: 250G
    ```

```
    image: phx.ocir.io/youracount/tt1814110:3
    imagePullSecret: sekret
    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.

See "TimesTenClassicSpecSpec" on page 11-3 for information on the `dbConfigMap` and the `dbSecret` attributes. See "Defining and creating the TimesTenClassic object" on page 4-2 for information on creating the TimesTenClassic object.

Using an init container

You can use an init container to get your metadata files into the `/ttconfig` directory of the TimesTen containers. An init container enables you to create your own scripts to populate the `/ttconfig` directory with the metadata files. For more information on init containers, see:

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

This example illustrates how to use an init container. It shows you where to specify the script that populates the `/ttconfig` directory (represented in **bold**). 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 will be automatically mounted at `/ttconfig` in your TimesTen containers. See `volumes` (represented in **bold**).

See "The TimesTenClassic object type" on page 11-1 and "Understanding the deployment process" on page 4-1 for details on the creating the TimesTenClassic object.

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: init1
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
  template:
    spec:
      imagePullSecrets:
      - name: sekret
      initContainers:
      - name: init1a
        image: phx.ocir.io/youraccount/tt1814110:3
        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: {}
```

Additional configuration options

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

- Persistent storage
- Resources specification for the tt and the daemonlog containers
- Pod location

Persistent storage

The Operator creates a Kubernetes StatefulSet object with the same name as the TimesTenClassic object. The StatefulSet associates one or more PersistentVolumeClaims with each Pod that it creates. This causes the associated volumes to be mounted in each Pod. These volumes persist across the instantiations of the Pod. If a Pod fails, the files that the Pod created in these volumes remain when Kubernetes creates a new Pod to replace the failed one.

When you create a TimesTenClassic object, you must specify `storageClassName` and you may specify `storageSize`. These attributes determine the characteristics of the PersistentVolumes.

The `storageClassName` must be one that is provided by the Kubernetes environment in which you are using. For example, in Oracle Kubernetes Environment (OKE), you may use `oci`.

In OKE, `50G` of storage is requested by default. Use the `storageSize` attribute to request a different size. The example in this section uses a `storageSize` and a `logStorageSize` that is greater than `50G`. `50G` of storage may be adequate for demonstration purposes, but in production environments, consider greater storage. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.

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/oracle/instances/instance1`.

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 for the transaction log files only. Use the `logStorageSize` attribute for this and control its placement by using the `logStorageClassName` attribute. This causes a second PersistentVolumeClaim 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.)

See "The TimesTenClassic object type" on page 11-1 and "Understanding the deployment process" on page 4-1 for details.

For example:

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

Resources specification for the tt and the daemonlog containers

You can specify specific resources requirements for the `tt` and the `daemonlog` containers. For example, you can tell Kubernetes that the `tt` container will require four CPUs and 20GB of RAM.

To do this, specify the `tt` and/or the `daemonlog` containers in the `containers` element in your TimesTenClassic object. In so doing, the Operator copies the resources datum from those containers verbatim into the definition of the container in the StatefulSet in which the Operator creates. Any other datum other than resources is ignored. If you do not specify resources for the `tt` container, the resources item is empty. There is no default.

The TimesTen memory and disk requirements are the same in Kubernetes as in any other environment. See "Storage provisioning for TimesTen" in the *Oracle TimesTen In-Memory Database Operations Guide* for information.

If you do not specify resources for the `daemonlog` container, there are defaults. (Note that the values are case sensitive. For example, `"20Mi"` is valid, but `"20mi"` is invalid.)

The defaults are:

- memory:`"20Mi"`
- cpu:`"100m"`

This example illustrates how to specify the resources requirements (represented in **bold**) for the `tt` and the `daemonlog` containers.

```
apiVersion:timesten.oracle.com/v1
kind:TimesTenClassic
metadata:
 name:sample
spec:
 ttspec:
   storageClassName:oci
   storageSize:250G
   image:...
   imagePullSecret:...
 template:
   spec:
     containers:
     - name:tt
       resources:
         requests:
           memory:"512Mi"
           cpu:   "1000m"
         limits:
           memory:"768Mi"
           cpu:   "2000m"
     - name:daemonlog
```

```
resources:
  requests:
    memory:"40Mi"
    cpu:   "200m"
```

Pod location

The Operator configures a replicated pair of TimesTen databases that can be used to provide high availability. However, in order to provide the appropriate level of high availability, you can control the placement of the TimesTen Pods in your Kubernetes cluster. For example, you may want to ensure that the TimesTen Pods are available in different availability zones, or are on different Kubernetes nodes.

Given that the requirements of every environment are different, the Operator does not attempt to control Pod placement. However, you can do this by specifying the `affinity` option in your TimesTenClassic object's spec's template. The Operator will pass the template to the StatefulSet that it creates.

See "The TimesTenClassic object type" on page 11-1 and "Understanding the deployment process" on page 4-1 for details.

For example:

```
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"
```

4

Deploying TimesTen 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 Chapter 11, "The TimesTenClassic Object Type" 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" on page 3-6 for information on ConfigMaps and Secrets. Also see "Using an init container" on page 3-12 and "Additional configuration options" on page 3-13 for other configuration options.) However, "Defining and creating the TimesTenClassic object" on page 4-2 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 "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 `250G` with the amount of storage that should be requested for each Pod to hold TimesTen. (This example assumes a production environment and uses 250G for storage. For demonstration purposes, you can use 50G of storage. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.)
   - `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`).
   - `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" on page 3-6 and "Example using one ConfigMap" on page 3-7 for information on ConfigMaps.

```
% vi sample.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    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:  2020-05-31T15:35:12Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:  2020-05-31T15:35:12Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:         18.1.4.11.0
    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:           18.1.4.11.0
    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 18.1.4.11.0
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-1 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-1 Release 18.1.4.11.0
  -     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
timestenclassic-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 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. TimesTen runs in the Pods as the `oracle` user. Once you have established a shell in the Pod, use the `su - oracle` command to

switch to the `oracle` user. After you switch to the `oracle` user, 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 and switch to the `oracle` user.

```
% kubectl exec -it sample-0 -c tt -- /usr/bin/su - oracle
```

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 `scott` user. Check that the `PermSize` value of `200` is correct. Check that the `scott.emp` table exists.

```
 % ttIsql sample

Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)

Command> connect adding "uid=scott;pwd=tiger" as scott;
Connection successful:
DSN=sample;UID=scott;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
scott: Command> tables;
  SCOTT.EMP
1 table found.
```

5

Using TimesTen Databases

This chapter explains how to use direct mode applications and Client/Server drivers to access and to use your TimesTen databases that reside in your Kubernetes cluster.

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 deployment. When configured, each Pod in your active standby pair runs two or more containers. One container runs TimesTen and the TimesTen agent and the other container(s) run whatever applications you choose. These applications that are running in your containers can then 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.

> **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.

The `.spec.template.spec.containers` attribute of your TimesTenClassic object can be specified to cause one or more containers to be created within each Pod in your active standby pair. These containers are created in addition to the `tt` container that runs TimesTen. The containers run the specified command in the container image.

This example illustrates how to include the `.spec.template.spec.containers` attribute in your TimesTenClassic object (represented in **bold**):

```
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: directmode
spec:
  ttspec:
    storageClassName: oci
```

```
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
    dbConfigMap:
    - directmode
  template:
    spec:
      containers:
      - name: yourapp
        image: phx.ocir.io/youraccount/yourapplication:1
        command: ["/bin/yourapp"]
      - name: anotherapp
        image: phx.ocir.io/youraccount/anotherapplication:1
        command: ["/bin/anotherapp"]
```

You can specify any other Kubernetes configuration for these containers just as you could in the `.spec.template.spec.containers` attribute of the Kubernetes StatefulSet object. For example, you could use the `resources` attribute to set CPU and memory limits for your containers. See "Resources specification for the tt and the daemonlog containers" on page 3-14 for information on the `resources` attribute. The Operator automatically adds appropriate mounts to the containers you specify. This enables your containers 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. See "Understanding how the Operator functions" on page 1-5 for information on TimesTen containers and specifically the `tt` container.

In particular:

- The name of the TimesTen users group is `oracle` with a group ID (GID) of `333`. For more information on the TimesTen users group, see "Understanding the TimesTen users group" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.
- TimesTen runs as the `oracle` operating system user (with UID of `333`). This `oracle` user is a member of the TimesTen group.
- You must configure your application containers to run your applications as a member of the TimesTen group with GID of `333`. Only members of this group can run TimesTen in direct mode.
- You can run your direct mode applications as a user with UID of `333`. However, this grants the application instance administrator permissions on the TimesTen instance. Alternatively, you can create a group with GID of `333` and then create a user whose primary or secondary group is that group, but with a UID that is not `333`. In the latter case, you can run your application as this user and you can use TimesTen in direct mode. You can then grant such a 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/oracle/instances/instance1`. The scripts to configure the TimesTen environment variables are located at `/tt/home/oracle/instances/instance1/bin/ttenv.*`.

- You must not modify any file that is located in the TimesTen instance. In addition, ensure you do not create any new files in the `$TIMESTEN_HOME` directory tree of the instance.
- There are two DSNs that have been configured. They are located in the `/tt/home/oracle/instances/instance1/conf/sys.odbc.ini` file. One DSN has the name of the TimesTenClassic object. The second is called `tt`. You can use either one. They are equivalent.
- You must not add entries to the `/tt/home/oracle/instances/instance1/conf/sys.odbc.ini` file. These files can be overwritten by the Operator. However, you can store your own DSN entries in the `$HOME/.odbc.ini` file located in your application container.
- You must 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.
- 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.

For example, if you chose to configure a `sys.odbc.ini` file, the contents of `sys.odbc.ini` would contain a client DSN definition that references the Pods that are running your TimesTen databases.

This example creates the `sample` DSN and references the `sample` TimesTenClassic object in the `default` 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 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 "Understanding the configuration metadata and the Kubernetes facilities" on page 3-1 for information on how to have the Operator create an initial user with `ADMIN` privileges.

In this example, use a connection string to connect to the `sample` database as the `scott` 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 `scott` user was created by the Operator and already exists in the `sample` database. After connecting, you can verify that the `scott.emp` table exists. (The Operator also previously created this table. See "schema.sql file" on page 3-5 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=scott;PWD=tiger";


Copyright (c) 1996, 2020, 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=scott;pwd=********";
Connection successful:
DSN=;TTC_SERVER=sample-0.sample.mynamespace.svc.cluster.local;
TTC_SERVER_DSN=sample;UID=scott;DATASTORE=/tt/home/oracle/datastore/sample;
DATABASECHARACTERSET=AL32UTF8;CONNECTIONCHARACTERSET=US7ASCII;PERMSIZE=200;
DDLREPLICATIONLEVEL=3;
(Default setting AutoCommit=1)
Command> tables;
  SCOTT.EMP
1 table found.
```

6

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
- Monitoring the health of the active standby pair of databases
- Understanding the BothDown state
- Understanding the ManualInterventionRequired state
- Bringing up one database
- Suspending the management of a TimesTenClassic object
- Locating the Operator
- Managing the TimesTen databases

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 Table 11–3, " 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 "Overview of the upgrade process" on page 10-1 for information on the upgrade process.

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" on page 6-5 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" on page 6-5 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" on page 6-7 and "Bringing up one database" on page 6-8 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 `reexamine` CRD syntax element 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" on page 6-7 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" on page 6-5 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" on page 6-5 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" on page 6-5 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" on page 11-3 for more information on the `pollingInterval` CRD syntax element 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" on page 6-4 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 `bothDownBehavior` CRD syntax element to determine what to do. See "TimesTenClassicSpecSpec" on page 11-3 for information.

If `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" on page 6-7 for information on the `ManualInterventionRequired` state.

If `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" on page 6-7 for details.
- 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" on page 6-3 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 `waitingForActiveTimeout` CRD syntax element. 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" on page 11-3 for more information on the `waitingForActiveTimeout` CRD syntax element.
- 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 `waitingForActiveTimeout`.
- If the database that is ahead cannot be loaded, the TimesTenClassic object enters the `ManualInterventionRequired` state. See "Understanding the ManualInterventionRequired state" on page 6-7 for details.

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" on page 6-8 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" on page 6-8 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" on page 6-8 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 `reexamine` CRD syntax element. Every `pollingInterval`, the Operator examines the value of `reexamine`. If the `reexamine` 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" on page 11-3 for more information on the `pollingInterval` and the `reexamine` CRD syntax elements.

The examination of the databases is performed exactly one time after you change the `reexamine` value. If the required conditions were not met, you may again attempt to meet them. You must then modify the `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 -- /usr/bin/su - oracle
```

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, 2021, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;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 2021

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

```
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 `reexamine` CRD syntax element.

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: phx.ocir.io/youraccount/tt1814110:3
    imagePullPolicy: Always
    imagePullSecret: sekret
    storageClassName: oci
    storageSize: 250G
    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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Reexamine:           April22reexamine1
    Stop Managing:       April21Stop1
    Storage Class Name:  oci
    Storage Size:        250G
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:         18.1.4.11.0
  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:            18.1.4.11.0
    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 -- /usr/bin/su - oracle
$ ttIsql sample

Copyright (c) 1996, 2021, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;
AutoCreate=0;PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=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 -- /usr/bin/su - oracle
% ttIsql sample

Copyright (c) 1996, 2021, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;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 `timestenclassic-operator`). This action prevents the Operator from interfering. See "Revert to manual control" on page 6-26 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 `stopManaging` CRD syntax element for this TimesTenClassic object. See "TimesTenClassicSpecSpec" on page 11-3 for more information on this element. The Operator examines the value of `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 `reexamine` CRD syntax element. See "Understanding the ManualInterventionRequired state" on page 6-7 for more information on the `ManualInterventionRequired` state and the `reexamine` CRD syntax element.

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

Suspend management of the TimesTenClassic object

This example illustrates how to use the `stopManaging` CRD syntax element 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
timestenclassic-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 `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 `stopManaging` in the file, then modify its value. It must be different than the current value.
   - If there is no line for `stopManaging` in the file, then add a line and specify a value.

   In this example, there is no `stopManaging` line. This example adds the `stopManaging` line and sets the value for `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: phx.ocir.io/youraccount/tt1814110:3
    imagePullPolicy: Always
    imagePullSecret: sekret
    storageClassName: oci
    storageSize: 250G
    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 `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" on page 6-8 to complete the process.

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 `timestenclassic-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 "Deploying the Operator" on page 2-6 for information on the deployment of the Operator.

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
timestenclassic-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 `oracle` user. Once you have established a shell in a Pod, use the `su - oracle` command to switch to the `oracle` user. This automatically configures your environment so you can run the TimesTen utilities.

> **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 -- /usr/bin/su - oracle
% ttIsql sample

Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
Command>
```

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" on page 6-29 and "Cleanup" on page A-19 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" on page 6-20 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" on page 6-22 section.
- If you are modifying general connection attributes, follow the procedure in the "Modifying general connection attributes" on page 6-24 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" on page 6-22 or the "Modifying general connection attributes" on page 6-24 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" on page 3-6 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.

```
5 kubectl describe configmap sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
adminUser:
----
scott/tiger
```

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

schema.sql:
----
create sequence scott.s;
create table scott.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: |
    scott/tiger
  db.ini: |
    PermSize=600
    TempSize=300
    ConnectionCharacterSet=AL32UTF8
    DatabaseCharacterSet=AL32UTF8
  schema.sql: |
    create sequence scott.s;
    create table scott.emp (id number not null primary key, name char (32));
kind: ConfigMap
metadata:
  creationTimestamp: "2020-09-15T19:23:59Z"
  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 scott.s;
create table scott.emp (id number not null primary key, name char (32));
```

```
adminUser:
----
scott/tiger

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" on page 6-22 section. If you are modifying only general connection attributes, proceed to the "Modifying general connection attributes" on page 6-24 section.

Modifying first connection attributes

If you have not modified the `db.ini` file, proceed to the "Manually edit the db.ini file" on page 6-20 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" on page 6-20 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 -- /usr/bin/su - oracle
% ttIsql sample
Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/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 "Failover" on page 10-27 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 -- /usr/bin/su - oracle
Last login: Sun Oct 11 15:50:29 UTC 2020 on pts/0
[oracle@sample-1 ~]$ ttIsql sample

Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/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 -- /usr/bin/su - oracle
% ttIsql sample

Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/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" on page 6-20 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" on page 6-22 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 -- /usr/bin/su - oracle
Last login: Sat Oct 10 22:43:26 UTC 2020 on pts/8
```

2. Verify the current directory (`/tt/home/oracle`).

```
% pwd
/tt/home/oracle
```

3. Navigate to the directory where the `sys.odbc.ini` file is located. The `sys.odbc.ini` file is located in the `/tt/home/oracle/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 18.1 Driver
tt=TimesTen 18.1 Driver

[sample]
Datastore=/tt/home/oracle/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, 2020, 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=oracle;DataStore=/tt/home/oracle/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 -- /usr/bin/su - oracle
```

```
Last login: Sat Oct 10 23:08:08 UTC 2020 on pts/0
```

2. Verify the current directory (`/tt/home/oracle`).

```
% pwd
/tt/home/oracle
```

3. Navigate to the directory where the `sys.odbc.ini` file is located. The `sys.odbc.ini` file is located in the `/tt/home/oracle/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 18.1 Driver
tt=TimesTen 18.1 Driver

[sample]
Datastore=/tt/home/oracle/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, 2020, 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=oracle;DataStore=/tt/home/oracle/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 `timestenclassic-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" on page 6-15 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
timestenclassic-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 "timestenclassic-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 -- /usr/bin/su - oracle
Last login: Wed Apr  8 14:30:45 UTC 2020 on pts/0
```

6. Run the `ttStatus` utility.

```
% ttStatus
TimesTen status report as of Wed Apr  8 14:36:31 2020

Daemon pid 183 port 6624 instance instance1
TimesTen server pid 190 started on port 6625
------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
Data store /tt/home/oracle/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 oracle
End of report
```

7. Run the `ttIsql` utility to connect to the `sample` database and perform various operations.

```
% ttIsql sample

Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
Command> describe scott.emp;

Table SCOTT.EMP:
  Columns:
   *ID                             NUMBER NOT NULL
    NAME                           CHAR (32)
```

```
1 table found.
(primary key columns are indicated with *)

Command> INSERT INTO scott.emp VALUES (1,'This is a test.');
1 row inserted.
Command> SELECT * FROM scott.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
```

7

Working with TimesTen Cache

This chapter describes how to use TimesTen Cache in your Kubernetes environment.

Topics include:

- Overview
- Creating the metadata files and the Kubernetes facility
- Creating the TimesTenClassic object
- Monitoring the deployment of the TimesTenClassic object
- Cleaning up the cache metadata on the Oracle Database

See Appendix B, "TimesTen Cache Example" for a complete example of configuring and using TimesTen Cache in your Kubernetes environment. The example also includes details on setting up the Oracle Database.

Overview

You can configure and then use TimesTen Cache in your Kubernetes environment. The TimesTen Operator provides these metadata files for this purpose:

- `cacheUser`: The `cacheUser` file is required. This file contains one line of the form:

  ```
  cacheUser/ttPassword/oraPassword
  ```

  where `cacheUser` is the user you want to designate as the TimesTen cache manager user. This user must have the same name as the user whom you designated as the cache administration user in the Oracle Database. The cache administration user must already exist in the Oracle Database. Specify `ttPassword` as the TimesTen password for the TimesTen `cacheUser` user (the TimesTen cache manager). The `oraPassword` is the Oracle Database password you specified when you created the `cacheUser` user in the Oracle Database.

  The Operator creates the `cacheUser` user with the `ttpassword` in the TimesTen database. This `cacheUser` user then serves as the cache manager user in your TimesTen database. (Note that you do not need to create this TimesTen user. The Operator does it for you.)

  See "Create the Oracle database users" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for information on the Oracle Database users. Also see "cacheUser" on page 3-3 for more information on the `cacheUser` metadata file.

- `cachegroups.sql`: The `cachegroups.sql` 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" on page 3-2 for more information on this file.

- `tnsnames.ora`: The `tnsnames.ora` file is required. This file defines Oracle Net Services to which applications connect. 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 file" on page 3-6 for more information on this file.
- `sqlnet.ora`: The `sqlnet.ora` file is not 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 file" on page 3-5 for information on this file.
- `db.ini`: The `db.ini` file is required. The contents of this file contain TimesTen connection attributes for your TimesTen databases, which will be included in TimesTen's `sys.odbc.ini` file. 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 file" on page 3-4 for more information on this file.
- `schema.sql`: The `schema.sql` 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 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" on page B-1 for more information on the schema users in the Oracle Database. Also see "schema.sql file" on page 3-5 for more information on the `schema.sql` 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 or replication, so ensure there are no cache definitions in this file.

The Operator looks for the presence of the `cacheUser` and the `cachegroups.sql` files in the `/ttconfig` directory of your TimesTen containers to determine if TimesTen Cache should be configured. If these files are present, the Operator, creates the TimesTen cache manager (from the contents of the `cacheUser` file) and starts the cache agent. The cache manager then uses the `ttIsql` utility to run the `cachegroups.sql` file.

The contents of the `cachegroups.sql` file is run 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.

Once your active standby pair of TimesTen databases are 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 databases (by deleting the TimesTenClassic object), the Operator automatically cleans up the Oracle Database metadata. If, however, you want to retain the Oracle Database metadata, specify the `cacheCleanUp` field in your TimesTenClassic object definition and set its value to `false`. See "Cleaning up the cache metadata on the

Oracle Database" on page 7-10 and see the cacheCleanUp entry in Table 11–3, " TimesTenClassicSpecSpec" for more information.

Creating the metadata files and the Kubernetes facility

You can include the cacheUser, cachegroups.sql, tnsnames.ora, sqlnet.ora, db.ini and the schema.sql 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" on page 3-6 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_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 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 user 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.

   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
   ```

4. 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;
```

5. Create the `tnsnames.ora` metadata file in this ConfigMap directory (`cm_cachetest`, in this example). See "tnsnames.ora file" on page 3-6 for more information on this file.

```
vi tnsnames.ora

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

6. Create the `sqlnet.ora` metadata file in this ConfigMap directory (`cm_cachetest`, in this example). See "sqlnet.ora file" on page 3-5 for more information on this file.

```
vi sqlnet.ora

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

7. 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" on page B-4 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
```

8. Create the `schema.sql` file in this ConfigMap directory (`cm_cachetest`, in this example). In this example, create the `oratt` user. (In this example, also assume this `oratt` user was previously created in the Oracle Database.) See "Create the Oracle Database users" on page B-1 for information on the `oratt` user in the Oracle Database.

```
vi schema.sql

create user oratt identified by ttpwd;
grant admin to oratt;
```

9. 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
```

10. 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.domain.com)))
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.domain.com)))

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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate. See the `storageSize` and

the logStorageSize entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.

- image: Replace phx.ocir.io/youraccount/tt1814110:3 with the location of the image registry (phx.ocir.io/youraccount) and the image containing TimesTen (tt1814110:3).
- 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: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
    imagePullPolicy: Always
    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 the 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:  2020-10-24T03:29:48Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:         18.1.4.11.0
    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:           18.1.4.11.0
    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 18.1.4.11.0
  -     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 18.1.4.11.0
  -     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 use TimesTen Cache in your Kubernetes environment.

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 autorefresh cache groups" in the *Oracle TimesTen Application-Tier Database Cache User's 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 11–3, " TimesTenClassicSpecSpec" for more information. Also see "The supported metadata files" on page 3-1 for information on the `cacheUser` and the `cachegroups.sql` files.

8

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 Linux 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 "Downloading TimesTen and the TimesTen Operator" on page 2-2 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/tt18.1.4.11.0/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 18.1 Release Notes are located here :
  'installation_dir/tt18.1.4.11.0/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" 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 "The supported metadata files" on page 3-1 for information on the `replicationWallet` and the `clientWallet` metadata files. Also see "Configuring TLS for replication" on page 8-3 and "Configuring TLS for Client/Server" on page 8-11 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" on page 8-1 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 "The supported metadata files" on page 3-1 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" on page 3-6 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" on page 8-1 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/oracle/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 "Understanding the configuration metadata and the Kubernetes facilities" on page 3-1 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 `scott` user with the `tiger` password.

   ```
   vi adminUser

   scott/tiger
   ```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

   ```
   vi schema.sql

   create sequence scott.s;
   create table scott.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:
----
scott/tiger

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

schema.sql:
----
create sequence scott.s;
create table scott.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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 the Table 11–3, " TimesTenClassicSpecSpec" 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 the Table 11–3, " TimesTenClassicSpecSpec" 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.
- `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`).
- `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: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
    imagePullPolicy: Always
    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:  2020-10-16T18:51:43Z
     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:                      phx.ocir.io/youraccount/tt1814110:3
    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:               250G
...
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 18.1.4.11.0
  -     StateChange  3m10s  ttclassic  Pod repltls-1 Daemon Up
  -     StateChange  2m3s   ttclassic  Pod repltls-0 Agent Up
  -     StateChange  2m3s   ttclassic  Pod repltls-0 Release 18.1.4.11.0
  -     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
```

```
timestenclassic-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 -- /usr/bin/su - oracle
```

3. Optional: From the shell in the active pod, verify the `cwallet.sso` file is located in the `/tt/home/oracle/replicationWallet` directory.

```
% ls /tt/home/oracle/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/oracle/instances/instance1/conf` directory.)

   In particular, note that:

   - `replication_wallet` is correctly set to `/tt/home/oracle/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/oracle/instances/instance1/conf/timesten.conf
admin_uid=333
admin_user=oracle
daemon_port=6624
group_name=oracle
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/oracle/replicationWallet
server_port=6625
show_date=1
timesten_release=18.1.4
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: 2020-10-16 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" on page 8-1 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/oracle/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 "Configuration on the server" 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 "The supported metadata files" on page 3-1 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" on page 3-6 for more information.

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" on page 8-1 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/oracle/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/oracle/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/oracle/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 "Configuration on the 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 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/oracle/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 scott user with the tiger password.

   ```
   vi adminUser
   ```

```
scott/tiger
```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

```
vi schema.sql

create sequence scott.s;
create table scott.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/oracle/csWallet
ciphersuites=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
encryption=required


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

adminUser:
----
scott/tiger
```

```
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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.
   - `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`).
   - `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: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
    imagePullPolicy: Always
    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:  2020-10-17T19:08:03Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
...
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 18.1.4.11.0
  -     StateChange  3m5s   ttclassic  Pod cstls-1 Agent Up
  -     StateChange  3m5s   ttclassic  Pod cstls-1 Release 18.1.4.11.0
  -     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" on page 8-1 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 -- /usr/bin/su - oracle
   ```

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" on page 8-1 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" on page 5-3 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" on page 8-18 for information.) For purposes of this example, the full path to the `clientWallet` directory is `/tt/home/oracle/clientWallet`. Therefore, in this example, `/tt/home/oracle/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 on the 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 on the 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 `scott` user. The `scott` 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=scott;PWD=tiger;
WALLET=tt/home/oracle/clientWallet;
CIPHERSUITES=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
ENCRYPTION=required";

Copyright (c) 1996, 2020, 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=scott;PWD=********;
WALLET=tt/home/oracle/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=scott;
DATASTORE=/tt/home/oracle/datastore/cstls;DATABASECHARACTERSET=AL32UTF8;
CONNECTIONCHARACTERSET=US7ASCII;AUTOCREATE=0;PERMSIZE=200;
DDLREPLICATIONLEVEL=3;FORCEDISCONNECTENABLED=1;(SERVER)ENCRYPTION=Required;
(SERVER)WALLET=file:/tt/home/oracle/csWallet;(client)Encryption=Required;
(client)Wallet=/tt/home/oracle/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.

9

Handling Failover and Recovery

This chapter illustrates how the 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" on page 6-3 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.

10

Performing Upgrades

This chapter describes the process for upgrading to a new patch (or patchset) of the Operator and of TimesTen. It also applies to downgrading. See "Overview of release numbers" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for information on TimesTen releases.

Topics include:

- Overview of the upgrade process
- Upgrading the Operator
- Upgrading TimesTen

Overview of the upgrade process

The upgrade process involves first upgrading the Operator and then upgrading TimesTen.

You must upgrade the Operator manually. This involves:

- Updating the `crd.yaml` and the `service_account.yaml` files.
- Building a new image that contains the new Operator.
- Switching to the new Operator

See "Upgrading the Operator" on page 10-2 for information on upgrading the Operator.

To upgrade TimesTen, you create a new container image that contains the new TimesTen release. You then update the TimesTenClassic objects associated with the active standby pairs by modifying the value of the TimesTenClassic objects' `image` CRD syntax element. See "TimesTenClassicSpecSpec" on page 11-3 for more information on the `image` CRD syntax element.

The Operator creates and associates a Kubernetes StatefulSet with each TimesTenClassic object. The StatefulSet causes and controls the creation of the Pods that run TimesTen.

When you modify the `image` CRD syntax element of a TimesTenClassic object, the Operator notices the change and modifies the corresponding `image` attribute(s) in the StatefulSet. This controls the future spawning of containers. If a running container terminates, then StatefulSet spawns one to replace it. This replaced container runs the newly specified image.

Similarly, for other containers you provided that are running in these Pods, the Operator examines the value of the each TimesTenClassic object's `image` value. This value controls which image your provided containers run. If you modify the `image`

value in the TimesTenClassic object, the corresponding attributes in the associated StatefulSet are modified by the Operator.

After the Operator propagates changes from the TimesTenClassic object to the associated StatefulSet, you must then determine if the Operator will perform the steps to upgrade TimesTen or if you will manually perform the steps. This is determined by the value you specified for the TimesTenClassic object's `imageUpgradeStrategy` CRD syntax element. A value of `auto` indicates the Operator should perform the upgrade whereas a value of `manual` indicates you will manually perform the upgrade. If you do not specify a value for `imageUpgradeStrategy`, the default is `auto`. See "TimesTenClassicSpecSpec" on page 11-3 for more information on the `imageUpgradeStrategy` CRD syntax element. For automated upgrades, (value of `imageUpgradeStrategy` is `auto`), you can also control the amount of time the Operator waits for a Pod to come up (after it has been deleted) by setting the `upgradeDownPodTimeout` value. See "TimesTenClassicSpecSpec" on page 11-3 for more information.

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 then spawns new containers to replace the original containers. These new containers run the newly specified TimesTen image. See "Using direct mode applications" on page 5-1 for more information on direct mode applications.

If there are failures in any steps of the upgrade process, the 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" on page 6-7 and "Bringing up one database" on page 6-8 for more information.

Upgrading the Operator

You must manually upgrade the Operator for both automated and manual upgrades. You can upgrade the current release of the TimesTen Operator to a new release while there are one or more TimesTenClassic objects running in your Kubernetes cluster and while the TimesTen databases that are associated with those TimesTenClassic objects are up and running.

The process of upgrading the Operator involves:

- Replacing the `crd.yaml` and the `service_account.yaml` files in the `/deploy` directory
- Building the new Operator image
- Updating the timestenclassic-operator Deployment

The Operator restarts, and the upgrade to the new Operator becomes effective. During the Operator upgrade, the TimesTen databases continue to run (even while there is no Operator managing them). After the Operator upgrade process is completed, the new Operator continues to manage the current TimesTenClassic objects in your cluster as well as the TimesTen databases associated with those TimesTenClassic objects.

These sections cover the steps for upgrading the Operator:

- Download the new release of the TimesTen Operator

- Replace the crd.yaml and the service_account.yaml files
- Build the new Operator image
- Review the current Operator
- Update the timestenclassic-operator Deployment

Download the new release of the TimesTen Operator

The new release of the TimesTen Operator is included in the new release of the TimesTen full distribution on Linux 64-bit. (In this example, the new release is `18.1.4.11.0`.)

Perform these steps to download the full distribution of TimesTen and then unpack the TimesTen Operator distribution that is embedded within it. Perform all steps from your Linux development host.

1. From the directory of your choice:
   - Create one subdirectory into which you will download the new TimesTen full distribution. For example, create the *new_installation_dir* subdirectory.
   - Create a second subdirectory into which you will unpack the new TimesTen Operator distribution. For example, create the *new_kube_files* subdirectory.

   ```
   % mkdir -p new_installation_dir
   % mkdir -p new_kube_files
   ```

2. Navigate to *new_installation_dir*.

   ```
   % cd new_installation_dir
   ```

   Download the TimesTen full distribution into this directory. In this example, download the `timesten1814110.server.linux8664.zip` file (the `18.1.4.11.0` full distribution for Linux 64-bit).

3. From the *new_installation_dir*, use the ZIP utility to unpack the TimesTen distribution.

   ```
   % unzip timesten1814110.server.linux8664.zip
   Archive:  /timesten/installation/timesten1814110.server.linux8664.zip
      creating: 18.1.4.11.0/
      creating: 18.1.4.11.0/ttoracle_home/
   ...
      creating: tt18.1.4.11.0/kubernetes/
   ...
   ```

   Note that the *new_installation_dir*`/tt18.1.4.11.0/kubernetes` directory is created. The `operator.zip` file is located in this directory. For example, this is a sample directory structure after unpacking the distribution (which can change from release to release):

   ```
   % pwd
   new_installation_dir/tt18.1.4.11.0
   % dir
   3rdparty  include     lib      oraclescripts  README.html  ttoracle_home
   bin       info        network  PERL           startup
   grid      kubernetes  nls      plsql          support
   ```

4. Navigate to the *new_kube_files* directory and unpack the `operator.zip` file into it. In this example, unpack the *new_installation_dir*`/tt18.1.4.11.0/kubernetes/operator.zip` file.

```
% cd new_kube_files
% unzip new_installation_dir/tt18.1.4.11.0/kubernetes/operator.zip
[...UNZIP OUTPUT...]
```

5. Review the directory structure. This example shows the most important subdirectories and files, which can change from release to release.

```
README.md
deploy/crd.yaml
deploy/operator.yaml
deploy/service_account.yaml
operator/Dockerfile
operator/timestenclassic-operator
ttimage/agent2
ttimage/.bashrc
ttimage/create1.sql
ttimage/create2.sql
ttimage/Dockerfile
ttimage/get1.sql
ttimage/pausecq.sql
ttimage/repcreate.sql
ttimage/repduplicate.sql
ttimage/runsql,sql
ttimage/starthost.pl
ttimage/.ttdotversion
ttimage/.ttdrop
```

Note: This directory tree must persist through the lifetime of the TimesTen Operator.

In addition, do not delete the TimesTen full distribution file (`timesten1814110.server.linux8664.zip`, in this example). You need to copy this file into the:

- `/operator` directory to build the new Operator image and push the image to the image registry. See "Build the new Operator image" on page 10-5 for details.
- `/ttimage` directory to build the new TimesTen image and push the image to the image registry. See "Build the new TimesTen image" on page 10-10 for details.

Replace the crd.yaml and the service_account.yaml files

You must replace the `crd.yaml` and the `service_account.yaml` files that reside in the `new_kube_files/ttdeploy` directory.

Note: Ensure you do not delete the `crd.yaml` file. Doing so deletes the TimesTenClassic objects along with the TimesTen databases associated with them.

Perform these steps:

1. Navigate to the `new_kube_files/deploy` directory and recreate the `crd.yaml` file.

```
% cd new_kube_files/deploy
% kubectl replace -f crd.yaml
```

```
customresourcedefinition.apiextensions.k8s.io/timestenclassics.timesten.
oracle.com replaced
```

2. While in the `new_kube_files/deploy` directory, recreate the Kubernetes service account in which the Operator runs. The Operator requires additional privileges to perform the upgrade procedure.

```
% kubectl replace -f service_account.yaml
role.rbac.authorization.k8s.io/timestenclassic-operator replaced
serviceaccount/timestenclassic-operator replaced
rolebinding.rbac.authorization.k8s.io/timestenclassic-operator replaced
```

You have successfully replaced the `crd.yaml` and the `service_account.yaml` files. You are now ready to build the Operator image.

Build the new Operator image

Before you can run the new Operator, you must build the new Operator image and push it to your image registry.

The files needed to build the new Operator image are provided in the *new_kube_files*`/operator` directory (part of the ZIP file you previously unpacked).

To build the new Operator image and push it to your registry, perform these steps:

1. Navigate to the *new_kube_files*`/operator` directory, and copy the TimesTen distribution into it. This example assumes you downloaded the `timesten1814110.server.linux8664.zip` distribution into the *new_installation_dir* directory. See "Download the new release of the TimesTen Operator" on page 10-3 for information. Then, verify the `timesten1814110.server.linux8664.zip` file is in the *new_kube_files*`/operator` directory.

```
% cd new_kube_files/operator
% cp new_installation_dir/timesten1814110.server.linux8664.zip .
% ls -a
Dockerfile
timesten1814110.server.linux8664.zip
timestenclassic-operator
```

2. Navigate to the *new_kube_files*`/operator` directory (if not already in this directory) and use the `docker` command to build and tag the new Operator image. When you are tagging the new Operator image, it is recommended that you tag the image with a release number. For example, you can use the naming convention: `ttclassic-operator:`*release* (where *release* is the release you wish to tag). In this example, `ttclassic-operator:3` is used to name the new Operator image (represented in **bold**).

```
% cd new_kube_files/operator
% docker build -t ttclassic-operator:3 .
Sending build context to Docker daemon  478.6MB
Step 1/7 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/7 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/7 : RUN yum -y install openssl unzip && /usr/sbin/useradd -d
/tt-operator -m -u 1001 -s /bin/nologin -U tt-operator
 ---> Using cache
 ---> e3f1427246ab
Step 4/7 : COPY --chown=tt-operator:tt-operator timestenclassic-operator
```

```
/usr/local/bin/timestenclassic-operator
 ---> Using cache
 ---> 6ccad53230f0
Step 5/7 : COPY --chown=tt-operator:tt-operator $TT_DISTRO /tt-operator/
$TT_DISTRO
 ---> 5cd31705485a
Step 6/7 : USER tt-operator
 ---> Running in 6a773ddac5dd
Removing intermediate container 6a773ddac5dd
 ---> 875ee38ebc75
Step 7/7 : ENTRYPOINT ["/usr/local/bin/timestenclassic-operator"]
 ---> Running in fed0f6c94c2f
Removing intermediate container fed0f6c94c2f
 ---> 10dde79e1617
Successfully built 10dde79e1617
Successfully tagged ttclassic-operator:3
```

3. Use the `docker` command to tag the new Operator image.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous step. (`ttclassic-operator:3` is represented in **bold** in this example.)

   ```
   % docker tag ttclassic-operator:3 phx.ocir.io/youraccount/ttclassic-operator:3
   ```

4. Use the `docker` command to push the new Operator image to your registry.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous steps. (`ttclassic-operator:3` is represented in **bold** in this example.)

   ```
   % docker push phx.ocir.io/youraccount/ttclassic-operator:3
   The push refers to repository [phx.ocir.io/youraccount/ttclassic-operator]
   46458e9fc890: Pushed
   471a399f0540: Pushed
   9e51a2b82af3: Pushed
   2f915858a916: Layer already exists
   3: digest:
   sha256:9b941f12e3d52298b9b38f7766ddcdfb1d011857a990ff01a8adafd32f3d3e8d size:
   1166
   ```

You successfully built the new Operator image and pushed it to your image registry.

Review the current Operator

This section provides the steps to review the current (running) Operator. These steps are not required.

1. Use the `kubectl get` command to ensure the current Operator is running (`timestenclassic-operator-66bd4bc88b-c8vf`, in this example, represented in **bold**).

   ```
   % kubectl get pods
   NAME                                        READY   STATUS    RESTARTS   AGE
   sample-0                                    2/2     Running   0          168m
   sample-1                                    2/2     Running   0          168m
   sample2-0                                   2/2     Running   0          158m
   sample2-1                                   2/2     Running   0          158m
   ```

```
timestenclassic-operator-66bd4bc88b-c8vfq   1/1     Running   0          3h5m
```

2. Use the `kubectl describe` command to review the current timestenclassic-operator Deployment. Note that the image for the Deployment is the original image (`phx.ocir.io/youraccount/ttclassic-operator:2`, in this example, represented in **bold**).

```
% kubectl describe deployment timestenclassic-operator
Name:                   timestenclassic-operator
Namespace:              mynamespace
CreationTimestamp:      Sun, 11 Apr 2021 13:40:36 +0000
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               name=timestenclassic-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=timestenclassic-operator
  Service Account:  timestenclassic-operator
  Containers:
   timestenclassic-operator:
    Image:      phx.ocir.io/youraccount/ttclassic-operator:2
    Port:       <none>
    Host Port:  <none>
    Command:
      timestenclassic-operator
    Environment:
      WATCH_NAMESPACE:   (v1:metadata.namespace)
      POD_NAME:          (v1:metadata.name)
      OPERATOR_NAME:    timestenclassic-operator
      GODEBUG:          x509ignoreCN=0
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   timestenclassic-operator-66bd4bc88b (1/1 replicas created)
Events:          <none>
```

3. Review the TimesTenClassic objects that are running in the Kubernetes cluster. There are two TimesTenClassic objects running (`sample` and `sample2`, in this example).

```
% kubectl get ttc
NAME      STATE    ACTIVE      AGE
sample    Normal   sample-0    3h8m
sample2   Normal   sample2-0   179m
```

Update the timestenclassic-operator Deployment

This section involves updating the current timestenclassic-operator Deployment to use the new Operator container image. After completing the steps in this section, the Operator is restarted and the upgrade to the new Operator becomes effective. If there is more than one Operator running, each Operator is restarted one at a time. This new

Operator will continue to manage the TimesTenClassic objects and the TimesTen databases associated with the those TimesTenClassic objects.

Perform these steps:

1. Navigate to the *new_kube_files*/deploy directory, and edit the operator.yaml file. This file is provided in the distribution that you previously unpacked. See "Download the new release of the TimesTen Operator" on page 10-3 for details.

   Update these fields represented in **bold** (in the operator.yaml file below):

   - replicas: 1

     Replace 1 with the number of copies of the Operator that you would like to run. 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 Kubernetes uses to pull images from your registry.

   - Replace the image line to reference the Operator image you just created. (phx.ocir.io/youraccount/ttclassic_operator:3, in this example).

```
% cd new_kube_files/deploy
% vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timestenclassic-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timestenclassic-operator
  template:
    metadata:
      labels:
        name: timestenclassic-operator
    spec:
      serviceAccountName: timestenclassic-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timestenclassic-operator
          image: phx.ocir.io/youraccount/ttclassic-operator:3
          command:
          - timestenclassic-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "timestenclassic-operator"
            - name: GODEBUG
              value: "x509ignoreCN=0"
```

2. From within the *new_kube_files*/deploy directory, use the kubectl replace command to update the timestenclassic-operator Deployment.

```
% kubectl replace -f operator.yaml
deployment.apps/timestenclassic-operator replaced
```

3. Use the kubectl get pods command to verify the new Operator is running (timestenclassic-operator-846cb5c97c-sbz22 in this example, represented in **bold**).

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
sample-0                                     2/2     Running   0          3h37m
sample-1                                     2/2     Running   0          3h37m
sample2-0                                    2/2     Running   0          3h28m
sample2-1                                    2/2     Running   0          3h28m
timestenclassic-operator-846cb5c97c-sbz22    1/1     Running   0          80s
```

4. Use the kubectl describe deployment command to view the new timetenclassic-operator Deployment. Note that the Operator is using the phx.ocir.io/youraccount/ttclassic-operator:3 image (represented in **bold**).

```
% kubectl describe deployment timestenclassic-operator
Name:                   timestenclassic-operator
Namespace:              mynamespace
CreationTimestamp:      Sun, 11 Apr 2021 13:40:36 +0000
Labels:                 name=timestenclassic-operator
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               name=timestenclassic-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=timestenclassic-operator
  Service Account:  timestenclassic-operator
  Containers:
   timestenclassic-operator:
    Image:      phx.ocir.io/youraccount/ttclassic-operator:3
    Port:       <none>
    Host Port:  <none>
    Command:
      timestenclassic-operator
    Environment:
      WATCH_NAMESPACE:   (v1:metadata.namespace)
      POD_NAME:          (v1:metadata.name)
      OPERATOR_NAME:     timestenclassic-operator
      GODEBUG:           x509ignoreCN=0
    Mounts:              <none>
  Volumes:               <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   timestenclassic-operator-846cb5c97c (1/1 replicas created)
Events:
  Type    Reason             Age    From                    Message
  ----    ------             ----   ----                    -------
```

```
  Normal  ScalingReplicaSet  4m19s  deployment-controller  Scaled up replica
set timestenclassic-operator-846cb5c97c to 1
  Normal  ScalingReplicaSet  3m51s  deployment-controller  Scaled down replica
set timestenclassic-operator-66bd4bc88b to 0
```

You have successfully updated the timestenclassic-operator Deployment. The new Operator automatically begins to manage any existing TimesTenClassic objects in your Kubernetes cluster.

Upgrading TimesTen

After you upgrade the Operator, you must upgrade your active standby pairs of TimesTen databases to a new patch of TimesTen. This involves building a new container image that contains the new TimesTen release and then modifying the `image` value of each of your TimesTenClassic objects to the name of this new container image.

When you modify the `image` value, the Operator notices the change and modifies the corresponding `image` attributes(s) in the StatefulSet.

What happens next is dependent on the value of the TimesTenClassic object's `imageUpgradeStrategy` element. See "TimesTenClassicSpecSpec" on page 11-3 for more information on `imageUpgradeStrategy`. If the value is:

- `auto` (or not specified): There are no additional steps you need to take. However, you can monitor the progress of the upgrade and then verify the upgrade was successful.
- `manual`: There are additional steps you must complete. These steps include upgrading the standby database and then performing the steps to fail over from the active database to the standby database. You can also verify the upgrade was successful.

These sections cover the steps necessary to upgrade each of your active standby pairs of TimesTen databases. For example purposes, there are two TimesTenClassic objects (`sample` and `sample2`) that require upgrading. The sample `TimesTenClassic` object (with an `imageUpgradeStrategy` value of `auto`) is upgraded first. Then the `sample2` TimesTenClassic object (with an `imageUpgradeStrategy` value of `manual`) is upgraded:

- Build the new TimesTen image
- Check the upgrade strategy for each TimesTenClassic object
- Complete one of the following depending on the upgrade strategy for the TimesTenClassic object. One of these procedures must be done for each TimesTenClassic object you wish to upgrade.
  - If `auto` (value of `imageUpgradeStrategy` is `auto` or not specified): Perform an automated upgrade
  - If `manual` (value of `imageUpgradeStrategy` is `manual`): Perform a manual upgrade
- Verify the active standby pair of databases are upgraded

Build the new TimesTen image

This section illustrates how to build TimesTen as a container image and then push the image to your image registry. The files that you need to build the new TimesTen image are provided in the *`new_kube_files`* directory tree. See "Download the new release of the TimesTen Operator" on page 10-3 for information.

To build the new TimesTen container image, perform these steps:

1. Navigate to the *new_kube_files*/ttimage directory, and copy the TimesTen distribution into it. This example assumes you downloaded the timesten1814110.server.linux8664.zip distribution into the *new_installation_dir* directory. See "Download the new release of the TimesTen Operator" on page 10-3 for information. Then, verify the timesten1814110.server.linux8664.zip file is in the *new_kube_files*/ttimage directory.

```
% cd new_kube_files/ttimage
% cp new_installation_dir/timesten1814110.server.linux8664.zip .
% ls *.zip
timesten1814110.server.linux8664.zip
```

2. Navigate to the *new_kube_files*/ttimage directory (if not already in this directory). Edit the Dockerfile, replacing timesten1814110.server.linux8664.zip with the name of your TimesTen full distribution. If your TimesTen distribution is timesten1814110.server.linux8664.zip, no modification is necessary. If not, the modification you need to make is represented in **bold**.

```
% cd new_kube_files/ttimage
% vi Dockerfile

# Copyright (c) 2019, 2021, Oracle and/or its affiliates.

FROM container-registry.oracle.com/os/oraclelinux:7

ARG TT_DISTRO=timesten1814110.server.linux8664.zip

RUN yum -y install tar gzip vim curl unzip libaio util-linux
RUN groupadd -g 333 oracle
RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle oracle
RUN install -d -m 0750 -o oracle -g oracle /home/oracle
COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop .ttdotversion agent2
create1.sql create2.sql get1.sql repcreate.sql repduplicate.sql runsql.sql
pausecg.sql /home/oracle/
# Uncomment the following line if you are using the optional non-root
installation procedure.
# USER 333
ENTRYPOINT "/home/oracle/starthost.pl"
```

3. Use the docker command to build the new TimesTen container image. Replace tt1814110:3 with a name of your choosing (represented in **bold**, in the docker build command below). Note that the output may change from release to release.

```
% docker build -t tt1814110:3 .

Sending build context to Docker daemon  445.8MB
Step 1/9 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/9 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/9 : RUN yum -y install tar gzip vim curl unzip libaio util-linux
 ---> Using cache
 ---> ac676b5376f3
Step 4/9 : RUN groupadd -g 333 oracle
 ---> Using cache
```

```
 ---> ce16920f085c
Step 5/9 : RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle
oracle
 ---> Using cache
 ---> 0319814aca1c
Step 6/9 : RUN install -d -m 0750 -o oracle -g oracle /home/oracle
 ---> Using cache
 ---> c8612b53398a
Step 7/9 : COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
 ---> 31cae98b71fd
Step 8/9 : COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop
.ttdotversion agent2 create1.sql create2.sql get1.sql repcreate.sql
repduplicate.sql runsql.sql pausecg.sql /home/oracle/
 ---> e50eb99c9b54
Step 9/9 : ENTRYPOINT "/home/oracle/starthost.pl"
 ---> Running in 0b41efd38837
Removing intermediate container 0b41efd38837
 ---> 171245e546d5
Successfully built 171245e546d5
Successfully tagged tt1814110:3
```

4. Use the `docker` command to tag the new TimesTen container image. Replace the following, represented in **bold**, in the `docker tag` command below.
   - `tt1814110:3` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

   ```
   % docker tag tt1814110:3 phx.ocir.io/youraccount/tt1814110:3
   ```

5. Use the `docker` command to push the new TimesTen container image to your registry. Replace the following, represented in **bold**, in the `docker push` command below.
   - `phx.ocir.io/youraccount` with the location of your image registry.
   - `tt1814110:3` with the name you chose previously.

   ```
   % docker push phx.ocir.io/youraccount/tt1814110:3

   The push refers to repository [phx.ocir.io/youraccount/tt1814110]
   97a0f250b2fe: Pushed
   650b003a3ad4: Pushed
   b8de51528854: Pushed
   62192d26e325: Pushed
   7dfe13e9b5a4: Pushed
   d8570fce965c: Pushed
   2f915858a916: Layer already exists
   3: digest:
   sha256:a6ac313394229eb2256d4a56fbcd8e2eda50ea2cc21991fa76f11701f2299710
   size: 1788
   ```

You successfully built the new TimesTen container image. It is pushed to your image registry. You are now ready to check the upgrade strategy for each TimesTenClassic object that you wish to upgrade.

Check the upgrade strategy for each TimesTenClassic object

The value of each of the TimesTenClassic object's `imageUpgradeStrategy` element determines the type of upgrade. If the value is `auto` (or not specified), the upgrade strategy is automated and the Operator does the upgrade for this TimesTenClassic

object. If the value is `manual`, the upgrade strategy is manual and you must manually perform the upgrade for the TimesTenClassic object.

This example shows you how to determine the `imageUpgradeStrategy` for the TimesTenClassic objects deployed in your Kubernetes cluster.

1. Review the TimesTenClassic objects that are running in the Kubernetes cluster. There are two TimesTenClassic objects running (`sample` and `sample2`, in this example).

```
% kubectl get ttc
NAME      STATE    ACTIVE      AGE
sample    Normal   sample-0    2d3h
sample2   Normal   sample2-0   2d3h
```

2. Use the `kubectl describe` command to show the `sample` TimesTenClassic object. Note that the `sample` TimesTenClassic object has `imageUpdateStrategy` set to `auto` (represented in **bold**). This indicates an automated upgrade strategy. Note also that `upgradeDownPodTimeout` is set to `900` (represented in **bold**).

```
% kubectl describe ttc sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-04-11T13:58:09Z
  Generation:          1
  Resource Version:    150145728
  Self Link:
  /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
  UID:                 f2a16dff-9acd-11eb-86a3-06b2b9dd76bc
Spec:
  Ttspec:
    Db Config Map:
      sample
    Image:                     phx.ocir.io/youraccount/tt181440:2
    Image Pull Policy:         Always
    Image Pull Secret:         sekret
    Image Upgrade Strategy:    auto
    Storage Class Name:        oci
    Storage Size:              250G
    Upgrade Down Pod Timeout:  900
...
```

3. Use the `kubectl describe` command to show the `sample2` TimesTenClassic object. Note that the `sample2` TimesTenClassic object has `imageUpdateStrategy` set to `manual` (represented in **bold**). This indicates a manual upgrade strategy.

```
% kubectl describe ttc sample2
Name:         sample2
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-04-11T14:07:20Z
  Generation:          1
```

```
  Resource Version:    150149654
  Self Link:
  /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample2
  UID:                 3af3e6fb-9acf-11eb-8286-6a1fd5dce8ff
Spec:
  Ttspec:
    Db Config Map:
      sample2
    Image:                   phx.ocir.io/youraccount/tt181440:2
    Image Pull Policy:       Always
    Image Pull Secret:       sekret
    Image Upgrade Strategy:  manual
    Storage Class Name:      oci
    Storage Size:            250G
...
```

You have successfully checked whether an automated or manual upgrade will be performed for each of the TimesTenClassic objects in your Kubernetes cluster. You are now ready to continue the upgrade process. This example upgrades the `sample` TimesTenClassic object first. Since the upgrade strategy for the `sample` TimesTenClassic object is `auto`, proceed to "Perform an automated upgrade" on page 10-14 to continue the upgrade procedure. (Note: You could upgrade the `sample2` TimesTenClassic object first. The order does not matter.)

Perform an automated upgrade

This section describes what you need to do for an automated upgrade. If you wish to do a manual upgrade, see "Perform a manual upgrade" on page 10-20 for details.

The automated upgrade process requires you to modify the image value of the TimesTenClassic object to reference the new TimesTen image (`18.1.4.11.0`, in this example). 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.

> **Note:** If you do an automated upgrade, your databases will be taken down, restarted, and failed over **immediately**. Do not perform this procedure at the busiest time of your production day. Applications will see short outages as a result of the upgrade procedure.

- Modify the TimesTenClassic object: automated upgrade
- Monitor the automated upgrade

Modify the TimesTenClassic object: automated upgrade

You must modify the TimesTenClassic object to reference the new TimesTen image. You do this by modifying the value of the `image` CRD syntax element(s) for the TimesTenClassic object. See "TimesTenClassicSpecSpec" on page 11-3 for more information on the `image` element.

The value of `imageUpgradeStrategy` for the `sample` TimesTenClassic object is `auto`, which indicates the Operator will perform the upgrade. After you edit the `image` value of the `sample` TimesTenClassic object to reference the new TimesTen image, the Operator notices the change, modifies the StatefulSet, and starts the automated upgrade process.

1. Use the `kubectl edit` command to edit the `sample` TimesTenClassic object, changing the `.spec.ttspec.image` attribute to reference the new TimesTen image (`phx.ocir.io/youraccount/tt1814110:3`, in this example, 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
...
  name: sample
...
spec:
  ttspec:
    dbConfigMap:
    - sample
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullPolicy: Always
    imagePullSecret: sekret
    imageUpgradeStrategy: manual
    storageClassName: oci
    storageSize: 250G
...
timestenclassic.timesten.oracle.com/sample edited
```

2. Use the `kubectl describe statefulset` command to verify that the Operator has modified the `sample` StatefulSet and replaced the image with the new image (`phx.ocir.io/youraccount/tt1814110:3`, in this example, represented in **bold**).

```
% kubectl describe statefulset sample
Name:               sample
Namespace:          mynamespace
CreationTimestamp:  Sun, 11 Apr 2021 13:58:10 +0000
Selector:           app=sample
Labels:             app=sample
Annotations:        <none>
Replicas:           2 desired | 2 total
Update Strategy:    OnDelete
Pods Status:        1 Running / 1 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=sample
  Init Containers:
   ttinit:
    Image:       phx.ocir.io/youraccount/tt1814110:3
    Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
    Command:
      perl
      /home/oracle/starthost.pl
    Environment:
      TIMESTEN_HOME:            /tt/home/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
```

```
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
      TT_INIT_CONTAINER:        1
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
  Containers:
   tt:
    Image:       phx.ocir.io/youraccount/tt1814110:3
    Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
    Command:
      perl
      /home/oracle/starthost.pl
    Environment:
      TIMESTEN_HOME:            /tt/home/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
   daemonlog:
    Image:      phx.ocir.io/youraccount/tt1814110:3
    Port:       <none>
    Host Port:  <none>
    Command:
      sh
      -c
      /bin/bash <<'EOF'
      while [ 1 ] ; do tail --follow=name
/tt/home/oracle/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/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
    Mounts:
      /tt from tt-persistent (rw)
  Volumes:
   tt-agent:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  ttf2a16dff-9acd-11eb-86a3-06b2b9dd76bc
    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:      250G
  Access Modes:  [ReadWriteOnce]
Events:
  Type    Reason            Age   From                    Message
  ----    ------            ----  ----                    -------
  Normal  SuccessfulCreate  7s    statefulset-controller  create Pod sample-1
in StatefulSet sample successful
```

You have successfully modified the `sample` TimesTenClassic object to use the new TimesTen image. You are now ready to monitor the automated upgrade process performed by the Operator. Proceed to "Monitor the automated upgrade" on page 10-17 to continue.

Monitor the automated upgrade

You can monitor the automated upgrade process performed by the Operator.

1. Use the `kubectl get` command to assess the state of the `sample` TimesTenClassic object.

   Note that the state is `StandbyDown` (represented in **bold**).

   ```
   % kubectl get ttc sample
   NAME     STATE         ACTIVE     AGE
   sample   StandbyDown   sample-1   2d5h
   ```

   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-1   2d5h
   ```

2. Use the `kubectl describe` command to observe how the Operator promoted the standby database (`sample-1`) to be the active. Note also that both the standby and the active databases have been upgraded to the new release of TimesTen (`18.1.4.11.0`, in this example).

   ```
   % kubectl describe ttc sample
   Name:         sample
   Namespace:    mynamespace
   Labels:       <none>
   Annotations:  <none>
   API Version:  timesten.oracle.com/v1
   Kind:         TimesTenClassic
   Metadata:
     Creation Timestamp:  2021-04-11T13:58:09Z
     Generation:          2
     Resource Version:    150178771
     Self Link:
   /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
     UID:                 f2a16dff-9acd-11eb-86a3-06b2b9dd76bc
   Spec:
     Ttspec:
       Db Config Map:
         sample
       Image:                     phx.ocir.io/youraccount/tt1814110:3
   ```

```
    Image Pull Policy:         Always
    Image Pull Secret:         sekret
    Image Upgrade Strategy:    auto
    Storage Class Name:        oci
    Storage Size:              250G
    Upgrade Down Pod Timeout:  900
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:                    67
  Last High Level State Switch:  1618341354
  Pod Status:
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:                          Loaded
      Db Id:                       35425
      Db Updatable:                No
    Initialized:                   true
    Last High Level State Switch:  ?
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1618341923
      Pod IP:               10.244.7.46
      Pod Phase:            Running
    Prev High Level State:  Healthy
    Prev Image:             phx.ocir.io/youraccount/tt181440:2
    Replication Status:
      Last Time Rep State Changed:  1618341175
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    STANDBY
    Times Ten Status:
      Daemon:          Up
      Instance:        Exists
      Release:         18.1.4.11.0
    Admin User File:   true
    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:       true
    Using Twosafe:     false
```

```
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:                         Loaded
      Db Id:                      35426
      Db Updatable:               Yes
    Initialized:                  true
    Last High Level State Switch:  ?
    Pod Status:
      Agent:               Up
      Last Time Reachable:  1618341923
      Pod IP:               10.244.6.25
      Pod Phase:            Running
    Prev High Level State:  Healthy
    Prev Image:             phx.ocir.io/youraccount/tt181440:2
    Replication Status:
      Last Time Rep State Changed:  1618340980
      Rep Agent:                    Running
      Rep Peer P State:             start
      Rep Scheme:                   Exists
      Rep State:                    ACTIVE
    Times Ten Status:
      Daemon:             Up
      Instance:           Exists
      Release:            18.1.4.11.0
    Admin User File:      true
    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:          true
    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
  Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     StateChange  15m    ttclassic  Image updated, automatic upgrade
started
  -     StateChange  15m    ttclassic  Deleted STANDBY pod sample-1 per
upgrade
  -     StateChange  15m    ttclassic  TimesTenClassic was Normal, now
ActiveTakeover
  -     StateChange  15m    ttclassic  TimesTenClassic was ActiveTakeover, now
StandbyDown
  -     StateChange  13m    ttclassic  Pod sample-1 Agent Up
```

```
  -     StateChange  13m    ttclassic  Pod sample-1 Release 18.1.4.11.0
  -     StateChange  13m    ttclassic  Pod sample-1 Instance Exists
  -     StateChange  13m    ttclassic  Pod sample-1 Daemon Up
  -     StateChange  13m    ttclassic  Pod sample-1 Database None
  -     StateChange  12m    ttclassic  Pod sample-1 RepState IDLE
  -     StateChange  12m    ttclassic  Pod sample-1 Database Loaded
  -     StateChange  12m    ttclassic  Pod sample-1 CacheAgent Not Running
  -     StateChange  12m    ttclassic  Pod sample-1 RepAgent Not Running
  -     StateChange  12m    ttclassic  Pod sample-1 RepScheme Exists
  -     StateChange  12m    ttclassic  Pod sample-1 Database Not Updatable
  -     StateChange  12m    ttclassic  Pod sample-1 RepAgent Running
  -     StateChange  12m    ttclassic  Pod sample-1 RepState STANDBY
  -     StateChange  12m    ttclassic  TimesTenClassic was StandbyDown, now
Normal
  -     StateChange  12m    ttclassic  Deleted ACTIVE pod sample-0 per upgrade
  -     StateChange  12m    ttclassic  TimesTenClassic was Normal, now
ActiveDown
  -     StateChange  12m    ttclassic  TimesTenClassic was ActiveDown, now
ActiveTakeover
  -     StateChange  12m    ttclassic  Pod sample-1 RepState ACTIVE
  -     StateChange  12m    ttclassic  Pod sample-1 Database Updatable
  -     StateChange  12m    ttclassic  TimesTenClassic was ActiveTakeover, now
StandbyDown
  -     StateChange  10m    ttclassic  Pod sample-0 Agent Up
  -     StateChange  10m    ttclassic  Pod sample-0 Instance Exists
  -     StateChange  10m    ttclassic  Pod sample-0 Daemon Up
  -     StateChange  10m    ttclassic  Pod sample-0 Database None
  -     StateChange  10m    ttclassic  Pod sample-0 Release 18.1.4.11.0
  -     StateChange  9m38s  ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  9m38s  ttclassic  Pod sample-0 Database Loaded
  -     StateChange  9m38s  ttclassic  Pod sample-0 Database Not Updatable
  -     StateChange  9m38s  ttclassic  Pod sample-0 RepAgent Not Running
  -     StateChange  9m38s  ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  9m38s  ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  9m32s  ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  9m32s  ttclassic  Pod sample-0 RepState STANDBY
  -     StateChange  9m32s  ttclassic  Upgrade of ACTIVE complete
  -     StateChange  9m32s  ttclassic  Upgrade completed in 385 secs
  -     StateChange  9m32s  ttclassic  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. If you need to upgrade additional TimesTenClassic objects, see "Perform an automated upgrade" on page 10-14 for an automated upgrade or "Perform a manual upgrade" on page 10-20 for a manual upgrade. If the upgrade process is complete for all TimesTenClassic objects, see "Verify the active standby pair of databases are upgraded" on page 10-31 to verify the upgrade of all TimesTenClassic objects that are running in your Kubernetes cluster.

This example now upgrades the `sample2` TimesTenClassic object. Recall that the value of the `imageUpgradePolicy` is `manual` for this TimesTenClassic object. Therefore, proceed to "Perform a manual upgrade" on page 10-20 to complete the upgrade for the `sample2` TimesTenClassic object.

Perform a manual upgrade

This section describes what you need to do for a manual upgrade. If you wish to do an automated upgrade, see "Perform an automated upgrade" on page 10-14 for details.

- Modify the TimesTenClassic object: manual upgrade
- Upgrade the standby database
- Failover

Modify the TimesTenClassic object: manual upgrade

You must modify the TimesTenClassic object to reference the new TimesTen image. You do this by modifying the value of the `image` CRD syntax element(s) for the TimesTenClassic object. See "TimesTenClassicSpecSpec" on page 11-3 for more information on the `image` element.

The value of `imageUpgradeStrategy` for the `sample2` TimesTenClassic object is `manual`, which indicates that you will manually perform the upgrade. After you edit the `image` value of the `sample2` TimesTenClassic object to reference the new TimesTen image, the Operator notices the change, and modifies the StatefulSet. The Operator does not restart the Pods. Rather, it upgrades the image that the Pods should be running. (The later sections entitled "Upgrade the standby database" on page 10-23 and "Failover" on page 10-27 detail the steps for restarting the Pods.)

1. Review the original `sample2.yaml` file. Note that `.spec.ttspec.image` references the `phx.ocir.io/youraccount/tt181440:2` image (represented in **bold**). Note also that the value of `imageUpgradeStrategy` is `manual` (represented in **bold**).

```
% cat sample2.yaml
apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
 name: sample2
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt181440:2
    imagePullSecret: sekret
    imagePullPolicy: Always
    dbConfigMap:
    - sample2
    imageUpgradeStrategy: manual
```

2. Use the `kubectl edit` command to edit the `sample2` TimesTenClassic object, changing the `.spec.ttspec.image` attribute to reference the new TimesTen image (`phx.ocir.io/youraccount/tt1814110:3`, in this example, represented in **bold**).

   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
...
spec:
  ttspec:
    dbConfigMap:
    - sample2
    image: phx.ocir.io/youraccount/tt1814110:3
```

```
    imagePullPolicy: Always
    imagePullSecret: sekret
    imageUpgradeStrategy: manual
    storageClassName: oci
    storageSize: 250G
...
timestenclassic.timesten.oracle.com/sample2 edited
```

3. Use the `kubectl describe statefulset` command to verify that the Operator has modified the `sample2` StatefulSet and replaced the image with the new image (`phx.ocir.io/youraccount/tt1814110:3`, in this example, represented in **bold**).

```
% kubectl describe statefulset sample2
Name:               sample2
Namespace:          mynamespace
CreationTimestamp:  Sun, 11 Apr 2021 14:07:21 +0000
Selector:           app=sample2
Labels:             app=sample2
Annotations:        <none>
Replicas:           2 desired | 2 total
Update Strategy:    OnDelete
Pods Status:        2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=sample2
  Init Containers:
   ttinit:
    Image:       phx.ocir.io/youraccount/tt1814110:3
    Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
    Command:
      perl
      /home/oracle/starthost.pl
    Environment:
      TIMESTEN_HOME:            /tt/home/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
      TT_INIT_CONTAINER:        1
    Mounts:
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
  Containers:
   tt:
    Image:       phx.ocir.io/youraccount/tt1814110:3
    Ports:       8443/TCP, 6624/TCP, 6625/TCP, 4444/TCP
    Host Ports:  0/TCP, 0/TCP, 0/TCP, 0/TCP
    Command:
      perl
      /home/oracle/starthost.pl
    Environment:
      TIMESTEN_HOME:            /tt/home/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
      TT_REPLICATION_TOPOLOGY:  activeStandbyPair
    Mounts:
```

```
      /tt from tt-persistent (rw)
      /ttagent from tt-agent (rw)
      /ttconfig from tt-config (rw)
   daemonlog:
    Image:      phx.ocir.io/youraccount/tt1814110:3
    Port:       <none>
    Host Port:  <none>
    Command:
      sh
      -c
      /bin/bash <<'EOF'
      while [ 1 ] ; do tail --follow=name
/tt/home/oracle/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/oracle/instances/instance1
      LD_LIBRARY_PATH:
/tt/home/oracle/instances/instance1/ttclasses/lib:/tt/home/oracle/instances/
instance1/install/lib:/tt/home/oracle/instances/instance1/install/
ttoracle_home/instantclient_11_2
    Mounts:
      /tt from tt-persistent (rw)
  Volumes:
   tt-agent:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  tt3af3e6fb-9acf-11eb-8286-6a1fd5dce8ff
    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:      250G
  Access Modes:  [ReadWriteOnce]
Events:          <none>
```

You have successfully modified the `sample2` TimesTenClassic object to use the new TimesTen image. You are now ready to continue the manual upgrade. Proceed to "Upgrade the standby database" on page 10-23 to continue this manual upgrade.

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 pods` command to review the Pods. The `sample2-0` and the `sample2-1` Pods are shown in this example.

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
sample2-0                                   2/2     Running   0          2d6h
sample2-1                                   2/2     Running   0          2d6h
```

2. 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`, (represented in **bold**) and the standby Pod is `sample2-1`.
   - Ensure the state for the TimesTenClassic object (`sample2`, in this example) is `Normal` (represented in **bold**).

```
% kubectl get ttc sample2
NAME     STATE    ACTIVE     AGE
sample2  Normal   sample2-0   2d6h
```

3. To upgrade the standby to the new TimesTen image, delete the standby Pod (`sample2-1`, in this example).

```
% kubectl delete pod sample2-1
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.

4. Use the `kubectl get` command to verify the standby is up and running and the state is `Normal`.

   Note that the state is `StandbyDown` (represented in **bold**).

```
% kubectl get ttc sample2
NAME     STATE         ACTIVE     AGE
sample2  StandbyDown   sample2-0   2d6h
```

   Wait a few minutes, then run the command again. Note that the state has changed to `Normal` (represented in **bold**).

```
% kubectl get ttc sample2
NAME     STATE    ACTIVE     AGE
sample2  Normal   sample2-0   2d6h
```

5. Use the `kubectl describe` command to further 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.

   In this example, note the following:

   - The image is upgraded to the new release (`phx.ocir.io/youraccount/tt1814110:3`, represented in **bold**).
   - The active database (`sample2-0`) is not upgraded to the new release. (Release is still `18.1.4.4.0`, represented in **bold**.)
   - The standby database (`sample2-1`) is upgraded to the new release. (`18.1.4.11.0`, represented in **bold**.)

```
% kubectl describe ttc sample2
Name:         sample2
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-04-11T14:07:20Z
  Generation:          2
  Resource Version:    150206835
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample2
  UID:                 3af3e6fb-9acf-11eb-8286-6a1fd5dce8ff
Spec:
  Ttspec:
    Db Config Map:
      sample2
    Image:                   phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:       Always
    Image Pull Secret:       sekret
    Image Upgrade Strategy:  manual
    Storage Class Name:      oci
    Storage Size:            250G
Status:
  Classic Upgrade Status:
    Active Start Time:           0
    Active Status:
    Image Update Pending:        true
    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:                   sample2-0
  High Level State:              Normal
  Last Event:                    41
  Last High Level State Switch:  1618347326
  Pod Status:
    Cache Status:
      Cache Agent:        Not Running
      Cache UID Pwd Set:  true
      N Cache Groups:     0
    Db Status:
      Db:                          Loaded
      Db Id:                       36351
      Db Updatable:                Yes
    Initialized:                   true
    Last High Level State Switch:  ?
    Pod Status:
      Agent:                Up
      Last Time Reachable:  1618347435
      Pod IP:               10.244.8.199
      Pod Phase:            Running
    Prev High Level State:  Healthy
    Prev Image:             phx.ocir.io/youraccount/tt181440:2
    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:         18.1.4.4.0
  Admin User File:   true
  Cache User File:   false
  Cg File:           false
  Disable Return:    false
  High Level State:  Healthy
  Intended State:    Active
  Local Commit:      false
  Name:              sample2-0
  Schema File:       true
  Using Twosafe:     false
  Cache Status:
    Cache Agent:        Not Running
    Cache UID Pwd Set:  true
    N Cache Groups:     0
  Db Status:
    Db:                         Loaded
    Db Id:                      36351
    Db Updatable:               No
  Initialized:                  true
  Last High Level State Switch:  ?
  Pod Status:
    Agent:               Up
    Last Time Reachable:  1618347435
    Pod IP:               10.244.5.156
    Pod Phase:            Running
  Prev High Level State:  Healthy
  Prev Image:             phx.ocir.io/youraccount/tt181440:2
  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:           18.1.4.11.0
  Admin User File:     true
  Cache User File:     false
  Cg File:             false
  Disable Return:      false
  High Level State:    Healthy
  Intended State:      Standby
  Local Commit:        false
  Name:                sample2-1
  Schema File:         true
  Using Twosafe:       false
Prev High Level State:  StandbyDown
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.mynamespace.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
  Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     StateChange  29m    ttclassic  Image updated, automatic upgrade
  disabled
  -     StateChange  4m35s  ttclassic  TimesTenClassic was Normal, now
  ActiveTakeover
  -     StateChange  4m29s  ttclassic  TimesTenClassic was ActiveTakeover, now
  StandbyDown
  -     StateChange  2m47s  ttclassic  Pod sample2-1 Agent Up
  -     StateChange  2m47s  ttclassic  Pod sample2-1 Release 18.1.4.11.0
  -     StateChange  2m47s  ttclassic  Pod sample2-1 Daemon Up
  -     StateChange  2m47s  ttclassic  Pod sample2-1 Database None
  -     StateChange  118s   ttclassic  Pod sample2-1 Database Loaded
  -     StateChange  118s   ttclassic  Pod sample2-1 Database Not Updatable
  -     StateChange  118s   ttclassic  Pod sample2-1 RepAgent Not Running
  -     StateChange  118s   ttclassic  Pod sample2-1 RepState IDLE
  -     StateChange  113s   ttclassic  Pod sample2-1 RepAgent Running
  -     StateChange  113s   ttclassic  Pod sample2-1 RepState STANDBY
  -     StateChange  113s   ttclassic  TimesTenClassic was StandbyDown, now
  Normal
```

You have successfully upgraded the standby database. You are now ready to fail over from the active database to the standby. See "Failover" on page 10-27 for details.

Failover

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* for information.)

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* for information.

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" on page 9-1 for more information of 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
pod "sample2-0" deleted
```

2. 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" on page 6-3 for information on the health and states of the active standby pair.

```
% kubectl describe ttc sample2
Name:         sample2
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-04-11T14:07:20Z
  Generation:          2
  Resource Version:    150214843
  Self Link:
/apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample2
  UID:                 3af3e6fb-9acf-11eb-8286-6a1fd5dce8ff
Spec:
  Ttspec:
    Db Config Map:
      sample2
    Image:                   phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:       Always
    Image Pull Secret:       sekret
    Image Upgrade Strategy:  manual
    Storage Class Name:      oci
    Storage Size:            250G
Status:
  Classic Upgrade Status:
    Active Start Time:           0
    Active Status:
    Image Update Pending:        true
    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:                   sample2-1
```

```
High Level State:              Normal
Last Event:                    57
Last High Level State Switch:  1618348960
Pod Status:
  Cache Status:
    Cache Agent:        Not Running
    Cache UID Pwd Set:  true
    N Cache Groups:     0
  Db Status:
    Db:                          Loaded
    Db Id:                       36623
    Db Updatable:                No
  Initialized:                   true
  Last High Level State Switch:  ?
  Pod Status:
    Agent:                Up
    Last Time Reachable:  1618349003
    Pod IP:               10.244.6.26
    Pod Phase:            Running
  Prev High Level State:  Healthy
  Prev Image:             phx.ocir.io/youraccount/tt181440:2
  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:         18.1.4.11.0
  Admin User File:   true
  Cache User File:   false
  Cg File:           false
  Disable Return:    false
  High Level State:  Healthy
  Intended State:    Standby
  Local Commit:      false
  Name:              sample2-0
  Schema File:       true
  Using Twosafe:     false
  Cache Status:
    Cache Agent:        Not Running
    Cache UID Pwd Set:  true
    N Cache Groups:     0
  Db Status:
    Db:                          Loaded
    Db Id:                       36624
    Db Updatable:                Yes
  Initialized:                   true
  Last High Level State Switch:  ?
  Pod Status:
    Agent:                Up
    Last Time Reachable:  1618349003
    Pod IP:               10.244.5.156
    Pod Phase:            Running
  Prev High Level State:  Healthy
  Prev Image:             phx.ocir.io/youraccount/tt181440:2
  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:            18.1.4.11.0
    Admin User File:      true
    Cache User File:      false
    Cg File:              false
    Disable Return:       false
    High Level State:     Healthy
    Intended State:       Active
    Local Commit:         false
    Name:                 sample2-1
    Schema File:          true
    Using Twosafe:        false
  Prev High Level State:  StandbyDown
  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.mynamespace.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
  Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     StateChange  55m    ttclassic  Image updated, automatic upgrade
disabled
  -     StateChange  30m    ttclassic  TimesTenClassic was Normal, now
ActiveTakeover
  -     StateChange  30m    ttclassic  TimesTenClassic was ActiveTakeover, now
StandbyDown
  -     StateChange  28m    ttclassic  Pod sample2-1 Agent Up
  -     StateChange  28m    ttclassic  Pod sample2-1 Release 18.1.4.11.0
  -     StateChange  28m    ttclassic  Pod sample2-1 Daemon Up
  -     StateChange  28m    ttclassic  Pod sample2-1 Database None
  -     StateChange  28m    ttclassic  Pod sample2-1 Database Loaded
  -     StateChange  28m    ttclassic  Pod sample2-1 Database Not Updatable
  -     StateChange  28m    ttclassic  Pod sample2-1 RepAgent Not Running
  -     StateChange  28m    ttclassic  Pod sample2-1 RepState IDLE
  -     StateChange  27m    ttclassic  Pod sample2-1 RepAgent Running
  -     StateChange  27m    ttclassic  Pod sample2-1 RepState STANDBY
  -     StateChange  27m    ttclassic  TimesTenClassic was StandbyDown, now
Normal
  -     StateChange  3m8s   ttclassic  TimesTenClassic was Normal, now
ActiveDown
  -     StateChange  3m1s   ttclassic  TimesTenClassic was ActiveDown, now
ActiveTakeover
  -     StateChange  3m1s   ttclassic  Pod sample2-1 RepState ACTIVE
  -     StateChange  3m1s   ttclassic  Pod sample2-1 Database Updatable
  -     StateChange  2m56s  ttclassic  TimesTenClassic was ActiveTakeover, now
StandbyDown
  -     StateChange  113s   ttclassic  Pod sample2-0 Agent Up
```

```
  -     StateChange  113s   ttclassic  Pod sample2-0 Release 18.1.4.11.0
  -     StateChange  113s   ttclassic  Pod sample2-0 Daemon Up
  -     StateChange  113s   ttclassic  Pod sample2-0 Database None
  -     StateChange  50s    ttclassic  Pod sample2-0 Database Loaded
  -     StateChange  50s    ttclassic  Pod sample2-0 Database Not Updatable
  -     StateChange  50s    ttclassic  Pod sample2-0 RepAgent Not Running
  -     StateChange  50s    ttclassic  Pod sample2-0 RepState IDLE
  -     StateChange  44s    ttclassic  Pod sample2-0 RepAgent Running
  -     StateChange  44s    ttclassic  Pod sample2-0 RepState STANDBY
  -     StateChange  44s    ttclassic  TimesTenClassic was StandbyDown, now
Normal
```

You have 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. If you need to upgrade additional TimesTenClassic objects, see "Perform an automated upgrade" on page 10-14 for an automated upgrade or "Perform a manual upgrade" on page 10-20 for a manual upgrade. If the upgrade process is complete for all TimesTenClassic objects, see "Verify the active standby pair of databases are upgraded" on page 10-31 to verify the upgrade of all TimesTenClassic objects that are running in your Kubernetes cluster.

In this example, the `sample` and the `sample2` TimesTenClassic objects are now upgraded. The upgrade process is complete. Proceed to "Verify the active standby pair of databases are upgraded" on page 10-31 to verify the TimesTenClassic objects are running successfully and have been upgraded.

Verify the active standby pair of databases are upgraded

After the upgrade process is complete for your TimesTenClassic objects, you can verify that each TimesTenClassic object is running successfully and that the active and the standby databases are running the new release of TimesTen.

1. Use the `kubectl get` command to verify the Pods are running.

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
sample-0                                    2/2     Running   0          7h1m
sample-1                                    2/2     Running   0          7h4m
sample2-0                                   2/2     Running   0          4h54m
sample2-1                                   2/2     Running   0          5h22m
timestenclassic-operator-846cb5c97c-sbz22   1/1     Running   0          2d8h
```

2. Use the `kubectl get` command to verify the state of each TimesTenClassic object is `Normal` (represented in **bold**, in this example).

```
% kubectl get ttc
NAME      STATE    ACTIVE      AGE
sample    Normal   sample-1    2d12h
sample2   Normal   sample2-1   2d12h
```

3. For the `sample` TimesTenClassic object, use the `kubectl exec -it` command to invoke a shell in the active Pod (`sample-1`, in this example). Then, run the `ttVersion` utility to verify the release is the new release. (`18.1.4.11.0`, in this example, represented in **bold**).

```
% kubectl exec -it sample-1 -c tt -- /usr/bin/su - oracle
% ttVersion
TimesTen Release 18.1.4.11.0 (64 bit Linux/x86_64) (instance1:6624)
2021-04-06T07:34:18Z
  Instance admin: oracle
```

```
  Instance home directory: /tt/home/oracle/instances/instance1
  Group owner: oracle
  Daemon home directory: /tt/home/oracle/instances/instance1/info
  PL/SQL enabled.
```

4. Now use the `kubectl exec -it` command to 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. (`18.1.4.11.0`, in this example, represented in **bold**).

```
% kubectl exec -it sample-0 -c tt -- /usr/bin/su - oracle
% ttVersion
TimesTen Release 18.1.4.11.0 (64 bit Linux/x86_64) (instance1:6624)
2021-04-06T07:34:18Z
  Instance admin: oracle
  Instance home directory: /tt/home/oracle/instances/instance1
  Group owner: oracle
  Daemon home directory: /tt/home/oracle/instances/instance1/info
  PL/SQL enabled.
```

5. For the `sample2` TimesTenClassic object, use the `kubectl exec -it` command to 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 (`18.1.4.11.0`, in this example).

```
% kubectl exec -it sample2-1 -c tt -- /usr/bin/su - oracle
% ttVersion
TimesTen Release 18.1.4.11.0 (64 bit Linux/x86_64) (instance1:6624)
2021-04-06T07:34:18Z
  Instance admin: oracle
  Instance home directory: /tt/home/oracle/instances/instance1
  Group owner: oracle
  Daemon home directory: /tt/home/oracle/instances/instance1/info
  PL/SQL enabled.
```

6. Now use the `kubectl exec -it` command to 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 (`18.1.4.11.0`, in this example).

```
% kubectl exec -it sample2-0 -c tt -- /usr/bin/su - oracle
% ttVersion
TimesTen Release 18.1.4.11.0 (64 bit Linux/x86_64) (instance1:6624)
2021-04-06T07:34:18Z
  Instance admin: oracle
  Instance home directory: /tt/home/oracle/instances/instance1
  Group owner: oracle
  Daemon home directory: /tt/home/oracle/instances/instance1/info
  PL/SQL enabled.
```

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.

11

The TimesTenClassic Object Type

This chapter describes the TimesTenClassic object type. You create objects of this type in order to create active standby pairs of TimesTen databases.

Topics:

- Overview of the TimesTenClassic object type
- The TimesTenClassic object type

Overview of the TimesTenClassic object type

The installation of the TimesTen Operator adds a new type of object to the Kubernetes cluster. You can create as many TimesTenClassic objects as you like. Each such object creates a pair of TimesTen databases, each running in a container, inside a Pod. Both Pods operate under the control of a StatefulSet.

The definition of the TimesTenClassic object type uses the same basic format as the formal Kubernetes documentation uses to define objects that are built-in to Kubernetes. Note that 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 Kuberenetes API reference documentation refers to a number of built-in Kubernetes types used in the definition of the TimesTenClassic object type, in particular the StatefulSet. In addition, since TimesTenClassic is basically a wrapper around a StatefulSet, its definition is particularly relevant. In particular, StatefulSetSpec is used as is. It describes the spec for the StatefulSet. It is how creators of StatefulSets express what they want the StatefulSet to look like. For more information, see:

`https://kubernetes.io/docs/reference/kubernetes-api/`

Note: All metadata is passed from the TimesTenClassic object to the StatefulSet.

The TimesTenClassic object type

The TimesTenClassic object type is defined using the following object definitions. These definitions are represented in table format. The first column includes the name of the field and the type. The second column provides a description.

- TimesTenClassic

- TimesTenClassicSpec
- TimesTenClassicSpecSpec
- TimesTenClassicStatus

TimesTenClassic

You create an object of type TimesTenClassic in order to create your active standby pair of TimesTen databases.

Table 11–1, " TimesTenClassic" shows the syntax for TimesTenClassic.

Table 11–1 TimesTenClassic

Field	Description
`apiVersion` string	`apiVersion` defines the versioned schema of this representation of an object. The value must be `timesten.oracle.com/v1`.
`kind` string	`kind` indicates the type of object (in this example, TimesTenClassic)
`metadata` `ObjectMeta`	`metadata` indicates the metadata about the object, such as its name. For information on `ObjectMeta`, see: `https://kubernetes.io/docs/reference/kubernetes-api/`
`spec` `TimesTenClassicSpec`	`spec` defines the desired configuration of TimesTen Pods and databases.
`status` `TimesTenClassicStatus`	`status` indicates the 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. See Table 11–1, " TimesTenClassic" for information.

Table 11–2, " TimesTenClassicSpec" shows the syntax for TimesTenClassicSpec.

Table 11–2 TimesTenClassicSpec

Field	Description
`ttspec` `TimesTenClassicSpecSpec`	`ttspec` defines the TimesTen specific attributes.
`template` `PodTemplateSpec`	`template` describes the Pod that is created if insufficient replicas are detected. Each Pod that is provisioned fulfills this template, but has a unique identity from the rest. There are two additional containers, named `tt` and `daemonlog`, that are automatically included in each Pod in addition to any specified here. TimesTen runs in the `tt` container. For information on `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/`

TimesTenClassicSpecSpec

TimesTenClassicSpecSpec appears in TimesTenClassicSpec. See Table 11–2, " TimesTenClassicSpec" for information.

Table 11–3, " TimesTenClassicSpecSpec" shows the syntax for TimesTenClassicSpecSpec.

Table 11–3 TimesTenClassicSpecSpec

Field	Description
`agentGetTimeout` integer	`agentGetTimeout` specifies the 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	`agentPostTimeout` specifies the 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. Note that `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	`agentTCPTimeout` specifies the 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	`agentTLSTimeout` specifies the 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.
`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" on page 6-4 for more information on the `BothDown` state.
`cacheCleanup` boolean	`cacheCleanup` specifies 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 "Cleaning up the cache metadata on the Oracle Database" on page 7-10 in this book for details. Also, see "Dropping Oracle Database objects used by autorefresh cache groups" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.
`daemonLogSidecar` boolean	`daemonLogSidecar` specifies whether a daemon log container is created in each TimesTen Pod. This container writes the TimesTen daemon logs (from `ttmesg.log`) to stdout, thus causing Kubernetes to log them. Valid values: ▪ `true` (or not specified): A daemon log container is created. ▪ `false`: A daemon log container is not created.
`dbConfigMap` array of strings	`dbConfigMap` specifies the names of one or more ConfigMaps that will be included in a ProjectedVolume. This ProjectedVolume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbConfigMap` or `dbSecret` (explained below), you must create the required files that need to be located in `/ttconfig` using other means. See "Populating the /ttconfig directory" on page 3-6 for details.

Table 11–3 (Cont.) TimesTenClassicSpecSpec

Field	Description
`dbSecret` array of strings	`dbSecret` specifies the names of one or more Secrets that will be included in a ProjectedVolume. This ProjectedVolume is mounted as `/ttconfig` in the TimesTen containers. If you do not specify `dbSecret` or `dbConfigMap` (explained above), you must create the required files that need to be located in `/ttconfig` using other means. See "Populating the /ttconfig directory" on page 3-6 for details.
`image` string	`image` defines the image containing TimesTen. There is no default. You must specify the name of the `image`.
`imagePullPolicy` string	`imagePullPolicy` 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	`imagePullSecret` defines the image pull secret that Kerbernetes should use to fetch the TimesTen image. There is no default. You must specify the name of the image pull secret.
`imageUpgradeStrategy` string	`imageUpgradeStrategy` specifies whether 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 Chapter 10, "Performing Upgrades" for information.
`logStorageClassName` string	`logStorageClassName` indicates the name of the storage class that is used to allocate PersistentVolumes to hold the TimesTen transaction logs. If you do not specify this field, the transaction logs are located in the PersistentVolumes defined by Kubernetes.
`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	`logStorageSize` is the amount of storage that should be requested for each Pod to hold the TimesTen transaction logs. See "Storage provisioning for TimesTen" in the *Oracle TimesTen In-Memory Database Operations Guide* for information on determining the amount of storage needed for the transaction log files. The default is `50G`. 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 `50G`. The examples in this book assume a production environment and use a value of `250G`.
`pollingInterval` integer	`pollingInterval` specifies 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`.

Table 11–3 (Cont.) TimesTenClassicSpecSpec

Field	Description
`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" on page 6-7 and "Bringing up one database" on page 6-8 for more information.

Table 11–3 (Cont.) TimesTenClassicSpecSpec

Field	Description

`repCreateStatement` string	The `repReturnServiceAttribute` and the `repStoreAttribute` syntax elements 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` syntax element 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`
	
	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` CRD syntax element.
	
	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" on page 6-3 for information on these states.
	
	Restrictions on the configuration:
	
	■ You must configure an active standby pair.
	■ You may not configure 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* for information.

Table 11–3 (Cont.) TimesTenClassicSpecSpec

Field	Description
`replicationCipherSuite` string	`replicationCipherSuite` specifies the encryption algorithm to be used by TimesTen replication. If not specified, replication traffic is not encrypted. You can specify either one of these values or both: ■ `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	`replicationSSLMandatory` specifies whether 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	`RepPort` specifies the TCP port to be used for replication. The default is `4444`.
`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.
`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" on page 6-7 and "Bringing up one database" on page 6-8 for more information.
`storageClassName` string	`storageClassName` indicates the name of the storage class that is used to allocate PersistentVolumes defined by Kubernetes. There is no default. You must specify the name of the storage class.

Table 11–3 (Cont.) TimesTenClassicSpecSpec

Field	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` element that you specify. You can optionally specify a label selector by using the `storageSelector` element. This label selector further filters the set of volumes. See: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#selector`
`storageSize` string	`storageSize` is the amount of storage that should be requested for each Pod to hold TimesTen. See "Storage provisioning for TimesTen" in the *Oracle TimesTen In-Memory Database Operations Guide* for information on determining the amount of storage needed for TimesTen. The default is `50G`. 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 `50G`. The examples in this book assume a production environment and use a value of `250G`.
`unreachableTimeout` integer	`unreachableTimeout` specifies the 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`.
`upgradeDownPodTimeout` integer	During an automated upgrade of TimesTen, the Operator deletes Pods. After deleting a Pod, the Operator waits up to the value of `upgradeDownPodTimeout` for the Pod to come back up. If the TimesTen agent located in the `tt` container of the Pod cannot be reached before this timeout, the TimesTenClassic object enter the `ManualInterventionRequired` state. The value is expressed in seconds. The default is `600`. A value of 0 indicates no timeout. The TimesTenClassic object waits forever and does not enter the `ManualInterventionRequired` state. See Chapter 10, "Performing Upgrades" for more information on the upgrade process.
`waitingForActiveTimeout` integerca	This setting specifies the 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). See "Monitoring the health of the active standby pair of databases" on page 6-3 for information on the `WaitingForActive` and the `ManualInterventionRequired` states.

TimesTenClassicStatus

TimesTenClassicStatus appears in TimesTenClassic. See "TimesTenClassic" on page 11-2 for information. This object type is a standard part of any CRD. The Operator stores various persistent information in TimesTenClassicStatus.

The status is displayed as part of the output of the `kubectl get` and `kubectl describe` commands.

Information in TimesTenClassicStatus includes:

- `awtBehindMb`: This field is only present 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. See "Overview of cache groups" in the

Oracle TimesTen Application-Tier Database Cache User's Guide for more information on AWT cache groups.

- High Level State of the Active Standby Pair: This is a string that describes the high level state of the active standby pair.
- Detailed state of TimesTen in each Pod, including:
  - 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.

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.

- Set up the environment
- 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

Set up the environment

Before starting the example, ensure you have:

- Completed the prerequisites. See "Prerequisites" on page 2-1 for information on the required prerequisites.

To set up the environment, perform these steps from your Linux development host:

- Download the TimesTen Operator
- Configure Kubernetes
- Deploy the TimesTenClassic CRD
- Build the Operator image
- Deploy the Operator
- Build the TimesTen image

Download the TimesTen Operator

Perform these steps to download the full distribution of TimesTen and then unpack the TimesTen Operator distribution that is embedded within it. Perform all steps from your Linux development host.

1. From the directory of your choice:
   - Create one subdirectory into which you will download the TimesTen full distribution. For example, create the *installation_dir* subdirectory. (The *installation_dir* directory is used in the remainder of this chapter.)
   - Create a second subdirectory into which you will unpack the TimesTen Operator distribution. For example, create the *kube_files* subdirectory. (This *kube_files* directory is used in the remainder of this chapter.)

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

   You are now ready to download and unpack the TimesTen full distribution.

2. Navigate to *installation_dir*.

   ```
   % cd installation_dir
   ```

   Download the TimesTen full distribution into this directory. As an example, download the `timesten1814110.server.linux8664.zip` file, (the `18.1.4.11.0` full distribution for Linux 64-bit).

3. From the *installation_dir*, use the ZIP utility to unpack the TimesTen distribution.

   ```
   % unzip timesten1814110.server.linux8664.zip
   Archive:  /timesten/installation/timesten1814110.server.linux8664.zip
      creating: tt18.1.4.11.0/
      creating: tt18.1.4.11.0/ttoracle_home/
   ...
      creating: tt18.1.4.11.0/kubernetes/
   ...
   ```

   You successfully unpacked the TimesTen full distribution.

   Note that the *installation_dir*`/tt18.1.4.11.0/kubernetes` directory is created. The `operator.zip` file is located in this directory. For example, this is a sample directory structure after unpacking the distribution.

   ```
   % pwd
   installation_dir/tt18.1.4.11.0
   % dir
   3rdparty  include     lib      oraclescripts  README.html  ttoracle_home
   bin       info        network  PERL           startup
   grid      kubernetes  nls      plsql          support
   ```

4. Navigate to the *kube_files* directory and unpack the `operator.zip` file into it. In this example, unpack the *installation_dir*`/tt18.1.4.11.0/kubernetes/operator.zip` file.

   ```
   % cd kube_files
   % unzip installation_dir/tt18.1.4.11.0/kubernetes/operator.zip
   [...UNZIP OUTPUT...]
   ```

   You successfully unpacked the *installation_dir*`/tt18.1.4.11.0/kubernetes/operator.zip` file into the *kube_files* directory.

5. Review the directory structure. Later in this chapter, you will modify some of the files in these subdirectories. This example shows the most important subdirectories and files, which can change from release to release.

```
README.md
deploy/crd.yaml
deploy/operator.yaml
deploy/service_account.yaml
operator/Dockerfile
operator/timestenclassic-operator
ttimage/agent2
ttimage/.bashrc
ttimage/create1.sql
ttimage/create2.sql
ttimage/Dockerfile
ttimage/get1.sql
ttimage/pausecq.sql
ttimage/repcreate.sql
ttimage/repduplicate.sql
ttimage/runsql,sql
ttimage/starthost.pl
ttimage/.ttdotversion
ttimage/.ttdrop
```

Note: This directory tree must persist through the lifetime of the TimesTen Operator.

In addition, do not delete the TimesTen full distribution file (`timesten1814110.server.linux8664.zip`, in this example). You need to copy this file into the:

- `/operator` directory to build the Operator image and push the image to the image registry. See "Build the Operator image" on page A-4 for details.
- `/ttimage` directory to build the TimesTen image and push the image to the image registry. See "Build the TimesTen image" on page A-7 for details.

You successfully downloaded and unpacked the TimesTen Operator distribution.

Configure Kubernetes

The Operator runs by using a Kubernetes *service account*. This service account needs permissions and privileges in your namespace. These permissions and privileges are granted through a *role*. The `service_account.yaml` file adds the service account and the role to your namespace, and grants the service account the privileges that are specified in the role. The `service_account.yaml` file is provided in the `operator.zip` file you previously unpacked.

Note: The provided role gives the `timestenclassic-operator` broad permissions within your namespace. Examine the permissions provided in the `service_account.yaml` file to see if the permissions need to be modified. If so, modify the permissions before running the commands in this example.

Perform these steps:

1. Navigate to the *kube_files*/deploy directory.

```
% cd kube_files/deploy
```

2. Create the service account.

```
% kubectl create -f service_account.yaml
role.rbac.authorization.k8s.io/timestenclassic-operator created
serviceaccount/timestenclassic-operator created
rolebinding.rbac.authorization.k8s.io/timestenclassic-operator created
```

The service_account.yaml file created the timestenclassic-operator service account and the timestenclassic-operator role in your namespace, and granted the service account the privileges specified in the role.

Deploy the TimesTenClassic CRD

Navigate to the *kube_files*/deploy directory, and then use the kubectl create command to create the TimesTenClassic customized resource definition (CRD) in your Kubernetes cluster.

```
% cd kube_files/deploy
% kubectl create -f crd.yaml
customresourcedefinition.apiextensions.k8s.io/
timestenclassics.timesten.oracle.com created
```

You successfully added the TimesTenClassic object type to your Kubernetes cluster.

Build the Operator image

Kubernetes Operators are Pods that run a customized image. Before you can run the Operator, you must build this image and push it to your image registry.

The files needed to create the image are provided in the *kube_files*/operator directory (part of the ZIP file you previously unpacked). In the *kube_files*/operator directory are the Dockerfile and the binaries needed to create the Operator image.

To build the Operator image and push it to your registry, perform these steps:

1. Navigate to the *kube_files*/operator directory, and copy the TimesTen distribution into it. This example assumes you downloaded the timesten1814110.server.linux8664.zip distribution into the *installation_dir* directory. See "Download the TimesTen Operator" on page A-2 for information. Then, verify the timesten1814110.server.linux8664.zip file is in the *kube_files*/operator directory.

```
% cd kube_files/operator
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls -a
Dockerfile
timesten1814110.server.linux8664.zip
timestenclassic-operator
```

2. Navigate to the *kube_files*/operator directory (if not already in this directory) and use the docker command to build the Operator image. You can choose any name for ttclassic-operator:3 (represented in **bold** in this example). Note that the output may change from release to release.

```
% cd kube_files/operator
% docker build -t ttclassic-operator:3 .
```

```
Sending build context to Docker daemon  478.6MB
Step 1/7 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/7 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/7 : RUN yum -y install openssl unzip && /usr/sbin/useradd -d
/tt-operator -m -u 1001 -s /bin/nologin -U tt-operator
 ---> Using cache
 ---> e3f1427246ab
Step 4/7 : COPY --chown=tt-operator:tt-operator timestenclassic-operator
/usr/local/bin/timestenclassic-operator
 ---> Using cache
 ---> 6ccad53230f0
Step 5/7 : COPY --chown=tt-operator:tt-operator $TT_DISTRO /tt-operator/
$TT_DISTRO
 ---> 5cd31705485a
Step 6/7 : USER tt-operator
 ---> Running in 6a773ddac5dd
Removing intermediate container 6a773ddac5dd
 ---> 875ee38ebc75
Step 7/7 : ENTRYPOINT ["/usr/local/bin/timestenclassic-operator"]
 ---> Running in fed0f6c94c2f
Removing intermediate container fed0f6c94c2f
 ---> 10dde79e1617
Successfully built 10dde79e1617
Successfully tagged ttclassic-operator:3
```

3. Use the `docker` command to tag the Operator image.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous step. (`ttclassic-operator` is represented in **bold** in this example.)

   ```
   % docker tag ttclassic-operator:3 phx.ocir.io/youraccount/ttclassic-operator:3
   ```

4. Use the `docker` command to push the Operator image to your registry.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry. (`phx.ocir.io/youraccount` is represented in **bold** in this example.)
   - Replace `ttclassic-operator:3` with the name you chose in the previous steps. (`ttclassic-operator:3` is represented in **bold** in this example.)

   ```
   % docker push phx.ocir.io/youraccount/ttclassic-operator:3
   The push refers to repository [phx.ocir.io/youraccount/ttclassic-operator]
   46458e9fc890: Pushed
   471a399f0540: Pushed
   9e51a2b82af3: Pushed
   2f915858a916: Layer already exists
   3: digest:
   sha256:9b941f12e3d52298b9b38f7766ddcdfb1d011857a990ff01a8adafd32f3d3e8d size:
   1166
   ```

You successfully built the Operator image and pushed it to your image registry.

Deploy the Operator

To deploy the Operator, you first customize it for your namespace and then deploy it. As a final step, you can verify the Operator is running. See "Deploying the Operator" on page 2-6 for information.

To customize the Operator for your namespace, navigate to the *kube_files*/deploy directory, and edit the operator.yaml file. This file is provided in the distribution that you previously unpacked. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for details.

1. Modify these fields represented in **bold** (in the operator.yaml file below):

   - replicas: 1

     Replace 1 with the number of copies of the Operator that you would like to run. 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 Kubernetes uses to pull images from your registry.
   - Replace phx.ocir.io/youraccount with the location of your image registry.
   - Replace ttclassic-operator:3 with the name you chose in the previous steps.

```
% cd kube_files/deploy
% vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timestenclassic-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timestenclassic-operator
  template:
    metadata:
      labels:
        name: timestenclassic-operator
    spec:
      serviceAccountName: timestenclassic-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timestenclassic-operator
          image: phx.ocir.io/youraccount/ttclassic-operator:3
          command:
          - timestenclassic-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
```

```
          value: "timestenclassic-operator"
        - name: GODEBUG
          value: "x509ignoreCN=0"
```

2. Use the `kubectl create` command to define the Operator to your namespace and to start the Operator.

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

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

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

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
timestenclassic-operator-f84766548-5bzch    1/1     Running   0          37s
```

   You verified that the Operator is running.

Build the TimesTen image

Before you can start TimesTen in your Kubernetes cluster, you must first package TimesTen as a container image and then push the image to your image registry. The files that you need to do this are provided in the *kube_files* directory tree. See "Building the TimesTen image" on page 2-8 for information.

To create the TimesTen container image, perform these steps:

1. Navigate to the *kube_files*/ttimage directory, and copy the TimesTen distribution into it. This example assumes you downloaded the `timesten1814110.server.linux8664.zip` distribution into the *installation_dir* directory. See "Download the TimesTen Operator" on page A-2 for information. Then, verify the `timesten1814110.server.linux8664.zip` file is in the *kube_files*/ttimage directory.

```
% cd kube_files/ttimage
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls *.zip
timesten1814110.server.linux8664.zip
```

2. Navigate to the *kube_files*/ttimage directory (if not already in this directory). Edit the `Dockerfile`, replacing `timesten1814110.server.linux8664.zip` with the name of your TimesTen full distribution. If your TimesTen distribution is `timesten1814110.server.linux8664.zip`, no modification is necessary. If not, the modification you need to make is represented in **bold**. Note: The TimesTen full distribution must be `18.1.4.11.0` or later.

```
% cd kube_files/ttimage
% vi Dockerfile

# Copyright (c) 2019, 2021, Oracle and/or its affiliates.

FROM container-registry.oracle.com/os/oraclelinux:7

ARG TT_DISTRO=timesten1814110.server.linux8664.zip

RUN yum -y install tar gzip vim curl unzip libaio util-linux
RUN groupadd -g 333 oracle
RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle oracle
```

```
RUN install -d -m 0750 -o oracle -g oracle /home/oracle
COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop .ttdotversion agent2
create1.sql create2.sql get1.sql repcreate.sql repduplicate.sql runsql.sql
pausecg.sql /home/oracle/
# Uncomment the following line if you are using the optional non-root
installation procedure.
# USER 333
ENTRYPOINT "/home/oracle/starthost.pl"
```

3. Use the `docker` command to build the TimesTen container image. Replace `tt1814110` with a name of your choosing (represented in **bold**, in the `docker build` command below). Note that the output may change from release to release.

```
% docker build -t tt1814110:3 .

Sending build context to Docker daemon  445.8MB
Step 1/9 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/9 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/9 : RUN yum -y install tar gzip vim curl unzip libaio util-linux
 ---> Using cache
 ---> ac676b5376f3
Step 4/9 : RUN groupadd -g 333 oracle
 ---> Using cache
 ---> ce16920f085c
Step 5/9 : RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle
oracle
 ---> Using cache
 ---> 0319814aca1c
Step 6/9 : RUN install -d -m 0750 -o oracle -g oracle /home/oracle
 ---> Using cache
 ---> c8612b53398a
Step 7/9 : COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
 ---> 31cae98b71fd
Step 8/9 : COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop
.ttdotversion agent2 create1.sql create2.sql get1.sql repcreate.sql
repduplicate.sql runsql.sql pausecg.sql /home/oracle/
 ---> e50eb99c9b54
Step 9/9 : ENTRYPOINT "/home/oracle/starthost.pl"
 ---> Running in 0b41efd38837
Removing intermediate container 0b41efd38837
 ---> 171245e546d5
Successfully built 171245e546d5
Successfully tagged tt1814110:3
```

4. Use the `docker` command to tag the TimesTen container image. Replace the following, represented in **bold**, in the `docker tag` command below.
   - `tt1814110:3` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

```
% docker tag tt1814110:3 phx.ocir.io/youraccount/tt1814110:3
```

5. Use the `docker` command to push the TimesTen container image to your registry. Replace the following, represented in **bold**, in the `docker push` command below.
   - `phx.ocir.io/youraccount` with the location of your image registry.

- `tt1814110:3` with the name you chose previously.

```
% docker push phx.ocir.io/youraccount/tt1814110:3

The push refers to repository [phx.ocir.io/youraccount/tt1814110]
97a0f250b2fe: Pushed
650b003a3ad4: Pushed
b8de51528854: Pushed
62192d26e325: Pushed
7dfe13e9b5a4: Pushed
d8570fce965c: Pushed
2f915858a916: Layer already exists
3: digest:
sha256:a6ac313394229eb2256d4a56fbcd8e2eda50ea2cc21991fa76f11701f2299710
size: 1788
```

You successfully built the TimesTen container image. It is pushed to your image registry.

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 "Understanding the configuration metadata and the Kubernetes facilities" on page 3-1 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 `scott` user with the `tiger` password.

   ```
   vi adminUser

   scott/tiger
   ```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

   ```
   vi schema.sql
   ```

```
create sequence scott.s;
create table scott.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:
   ----
   scott/tiger

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

   schema.sql:
   ----
   create sequence scott.s;
   create table scott.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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.
   - `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`).
   - `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" on page 3-6 and "Example using one ConfigMap" on page 3-7 for information on ConfigMaps.

```
% vi sample.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    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:  2020-05-31T15:35:12Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:  2020-05-31T15:35:12Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:         18.1.4.11.0
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:           18.1.4.11.0
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 18.1.4.11.0
  -     StateChange  3m26s  ttclassic  Pod sample-0 Daemon Down
  -     StateChange  3m26s  ttclassic  Pod sample-1 Agent Up
  -     StateChange  3m26s  ttclassic  Pod sample-1 Release 18.1.4.11.0
  -     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
timestenclassic-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. TimesTen runs in the Pods as the `oracle` user. Once you have established a shell in the Pod, use the `su - oracle` command to switch to the `oracle` user. After you switch to the `oracle` user, 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 and switch to the `oracle` user

```
% kubectl exec -it sample-0 -c tt -- /usr/bin/su - oracle
```

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 `scott` user. Check that the `PermSize` value of `200` is correct. Check that the `scott.emp` table exists.

```
 % ttIsql sample
```

```
Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)

Command> connect adding "uid=scott;pwd=tiger" as scott;
Connection successful:
DSN=sample;UID=scott;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=200;
DDLReplicationLevel=3;
(Default setting AutoCommit=1)
scott: Command> tables;
  SCOTT.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" on page 6-3 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
timestenclassic-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 "timestenclassic-operator" deleted
```

B

TimesTen Cache Example

This appendix provides a working example for using TimesTen Cache 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
- Creating the metadata files and the Kubernetes facility
- Creating the TimesTenClassic object
- Monitoring the deployment of the TimesTenClassic object
- Verifying that TimesTen Cache is configured correctly
- Performing operations on the cache group tables
- Cleaning up the cache metadata on the Oracle Database

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
- Grant privileges to the cache administration user
- Create the Oracle Database tables to be cached

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" in the *Oracle TimesTen Application-Tier Database Cache User's 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" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for information.

   This example creates the `cachetablespace2` tablespace.

```
% sqlplus sys/syspwd@oracache as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 23 22:10:20 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options

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. See "cacheUser" on page 3-3 and see "Creating the metadata files and the Kubernetes facility" on page 7-3 for details on 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 the cache administration user and the cache manager user" in the *Oracle TimesTen Application-Tier Database Cache User's 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 "Downloading TimesTen and the TimesTen Operator" on page 2-2 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 Application-Tier Database Cache User's Guide* for more information.

   ```
   % sqlplus sys/syspwd@oracache as sysdba

   SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 23 22:10:20 2020

   Copyright (c) 1982, 2014, Oracle.  All rights reserved.


   Connected to:
   Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
   With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
   options

   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" on page B-1 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*Plus: Release 12.1.0.2.0 Production on Fri Oct 23 22:10:20 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options

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" on page B-6 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" on page B-1 for information on the cache administration user in the Oracle Database. Also see "cacheUser" on page 3-3 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" on page 3-2 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 file" on page 3-6 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 file" on page 3-5 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 file" on page 3-4 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" on page B-1 for more information on the schema users in the Oracle Database. Also see "schema.sql file" on page 3-5 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 file" on page 3-2 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" on page 3-5 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" on page 3-6 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 `scott` user with the `tiger` password.

```
vi adminUser

scott/tiger
```

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" on page B-4 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" on page B-1 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" on page B-1 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.domain.com)))
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.domain.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.domain.com)))
OraCache =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = database.myhost.svc.cluster.local)
     (PORT = 1521))
   (CONNECT_DATA =
     (SERVER = DEDICATED)
     (SERVICE_NAME = OraCache.my.domain.com)))

adminUser:
----
scott/tiger

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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 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" on page 11-3 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate.
   - `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`).
   - `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: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    imagePullSecret: sekret
    imagePullPolicy: Always
    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:  2020-10-24T03:29:48Z
  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:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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:         18.1.4.11.0
    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:          18.1.4.11.0
    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 18.1.4.11.0
  -     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 18.1.4.11.0
  -     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
   timestenclassic-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 -- /usr/bin/su - oracle
   ```

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, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/cachetest;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;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" on page B-6 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*Plus: Release 12.1.0.2.0 Production on Fri Oct 23 21:57:42 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options

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 -- /usr/bin/su - oracle
```

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, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/cachetest;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;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 -- /usr/bin/su - oracle
   ```

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, 2020, 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/oracle/datastore/cachetest;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;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*Plus: Release 12.1.0.2.0 Production on Fri Oct 23 21:57:42 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options

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 autorefresh cache groups" in the *Oracle TimesTen Application-Tier Database Cache User's 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 11–3, " TimesTenClassicSpecSpec" for more information. Also see "The supported metadata files" on page 3-1 for information on the `cacheUser` and the `cachegroups.sql` files.

C

Run Containers as Non-Root

The Dockerfiles, as shipped, contain one Perl script that runs as root. This Perl script performs an initial configuration of the `tt` container in each Pod. It then switches from `root` to the `oracle` user. If you do not want any process in any container to run as root, you can use the procedure in this appendix.

Note: TimesTen (including the TimesTen daemons, the TimesTen agent, and the Kubernetes Operator) does not run as root.

The appendix shows a complete example.

The overview describes the requirements for configuring the TimesTen containers to run as non-root. The remaining sections show the standard setup procedures for configuring your active standby pairs of TimesTen databases. The final section shows you how to verify the TimesTen containers are running as non-root.

Topics:

- Overview
- Set up the environment
- Create the ConfigMap object
- Create the TimesTenClassic object
- Monitor deployment
- Verify the TimesTen container runs as non-root

Overview

You can configure your containers to run as non-root. This involves two additional steps from the standard setup procedures:

1. Uncomment `USER 333` in the `Dockerfile` located in the `/ttimage` directory. See "Build the TimesTen image" on page C-7 for details.
2. Include `.template.spec.securityContext` information in the YAML file of your TimesTenClassic object. See "Create the TimesTenClassic object" on page C-11 for details.

Set up the environment

Before starting the example, ensure you have:

- Completed the prerequisites. See "Prerequisites" on page 2-1 for information on the required prerequisites.

To set up the environment, perform these steps from your Linux development host:

- Download the TimesTen Operator
- Configure Kubernetes
- Deploy the TimesTenClassic CRD
- Build the Operator image
- Deploy the Operator
- Build the TimesTen image

Download the TimesTen Operator

Perform these steps to download the full distribution of TimesTen and then unpack the TimesTen Operator distribution that is embedded within it. Perform all steps from your Linux development host.

1. From the directory of your choice:
   - Create one subdirectory into which you will download the TimesTen full distribution. For example, create the *installation_dir* subdirectory. (The *installation_dir* directory is used in the remainder of this chapter.)
   - Create a second subdirectory into which you will unpack the TimesTen Operator distribution. For example, create the *kube_files* subdirectory. (This *kube_files* directory is used in the remainder of this chapter.)

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

   You are now ready to download and unpack the TimesTen full distribution.

2. Navigate to *installation_dir*.

   ```
   % cd installation_dir
   ```

   Download the TimesTen full distribution into this directory. As an example, download the `timesten1814110.server.linux8664.zip` file, (the `18.1.4.11.0` full distribution for Linux 64-bit).

3. From the *installation_dir*, use the ZIP utility to unpack the TimesTen distribution.

   ```
   % unzip timesten1814110.server.linux8664.zip
   Archive:  /timesten/installation/timesten1814110.server.linux8664.zip
      creating: tt18.1.4.11.0/
      creating: tt18.1.4.11.0/ttoracle_home/
   ...
      creating: tt18.1.4.11.0/kubernetes/
   ...
   ```

   You successfully unpacked the TimesTen full distribution.

   Note that the *installation_dir*`/tt18.1.4.11.0/kubernetes` directory is created. The `operator.zip` file is located in this directory. For example, this is a sample directory structure after unpacking the distribution:

   ```
   % pwd
   installation_dir/tt18.1.4.11.0
   % dir
   3rdparty  include     lib      oraclescripts  README.html  ttoracle_home
   bin       info        network  PERL           startup
   ```

```
grid      kubernetes  nls      plsql          support
```

4. Navigate to the *kube_files* directory and unpack the `operator.zip` file into it. In this example, unpack the *installation_dir*`/tt18.1.4.11.0/kubernetes/operator.zip` file.

   ```
   % cd kube_files
   % unzip installation_dir/tt18.1.4.11.0/kubernetes/operator.zip
   [...UNZIP OUTPUT...]
   ```

   You successfully unpacked the *installation_dir*`/tt18.1.4.11.0/kubernetes/operator.zip` file into the *kube_files* directory.

5. Review the directory structure. Later in this chapter, you will modify some of the files in these subdirectories. This example shows the most important subdirectories and files, which can change from release to release.

   ```
   README.md
   deploy/crd.yaml
   deploy/operator.yaml
   deploy/service_account.yaml
   operator/Dockerfile
   operator/timestenclassic-operator
   ttimage/agent2
   ttimage/.bashrc
   ttimage/create1.sql
   ttimage/create2.sql
   ttimage/Dockerfile
   ttimage/get1.sql
   ttimage/pausecq.sql
   ttimage/repcreate.sql
   ttimage/repduplicate.sql
   ttimage/runsql,sql
   ttimage/starthost.pl
   ttimage/.ttdotversion
   ttimage/.ttdrop
   ```

Note: This directory tree must persist through the lifetime of the TimesTen Operator.

In addition, do not delete the TimesTen full distribution file (`timesten1814110.server.linux8664.zip`, in this example). You need to copy this file into the:

- `/operator` directory to build the Operator image and push the image to the image registry. See "Build the Operator image" on page C-4 for details.
- `/ttimage` directory to build the TimesTen image and push the image to the image registry. See "Build the TimesTen image" on page C-7 for details.

You successfully downloaded and unpacked the TimesTen Operator distribution.

Configure Kubernetes

The Operator runs by using a Kubernetes *service account*. This service account needs permissions and privileges in your namespace. These permissions and privileges are

granted through a *role*. The `service_account.yaml` file adds the service account and the role to your namespace, and grants the service account the privileges that are specified in the role. The `service_account.yaml` file is provided in the `operator.zip` file you previously unpacked.

> **Note:** The provided role gives the `timestenclassic-operator` broad permissions within your namespace. Examine the permissions provided in the `service_account.yaml` file to see if the permissions need to be modified. If so, modify the permissions before running the commands in this example.

Perform these steps:

1. Navigate to the *kube_files*`/deploy` directory.

   ```
   % cd kube_files/deploy
   ```

2. Create the service account.

   ```
   % kubectl create -f service_account.yaml
   role.rbac.authorization.k8s.io/timestenclassic-operator created
   serviceaccount/timestenclassic-operator created
   rolebinding.rbac.authorization.k8s.io/timestenclassic-operator created
   ```

The `service_account.yaml` file created the `timestenclassic-operator` service account and the `timestenclassic-operator` role in your namespace, and granted the service account the privileges specified in the role.

Deploy the TimesTenClassic CRD

Navigate to the *kube_files*`/deploy` directory, and then use the `kubectl create` command to create the TimesTenClassic customized resource definition (CRD) in your Kubernetes cluster.

```
% cd kube_files/deploy
% kubectl create -f crd.yaml
customresourcedefinition.apiextensions.k8s.io/
timestenclassics.timesten.oracle.com created
```

You successfully added the TimesTenClassic object type to your Kubernetes cluster.

Build the Operator image

Kubernetes Operators are Pods that run a customized image. Before you can run the Operator, you must build this image and push it to your image registry.

The files needed to create the image are provided in the *kube_files*`/operator` directory (part of the ZIP file you previously unpacked). In the *kube_files*`/operator` directory are the `Dockerfile` and the binaries needed to create the Operator image.

To build the Operator image and push it to your registry, perform these steps:

1. Navigate to the *kube_files*`/operator` directory, and copy the TimesTen distribution into it. This example assumes you downloaded the `timesten1814110.server.linux8664.zip` distribution into the *installation_dir* directory. See "Download the TimesTen Operator" on page C-2 for information. Then, verify the `timesten1814110.server.linux8664.zip` file is in the *kube_files*`/operator` directory.

```
% cd kube_files/operator
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls -a
Dockerfile
timesten1814110.server.linux8664.zip
timestenclassic-operator
```

2. Navigate to the *kube_files*/operator directory (if not already in this directory) and use the docker command to build the Operator image. You can choose any name for ttclassic-operator:2 (represented in **bold** in this example). Note that the output may change from release to release.

```
% cd kube_files/operator
% docker build -t ttclassic-operator:3 .
Sending build context to Docker daemon  478.6MB
Step 1/7 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/7 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/7 : RUN yum -y install openssl unzip && /usr/sbin/useradd -d
/tt-operator -m -u 1001 -s /bin/nologin -U tt-operator
 ---> Using cache
 ---> e3f1427246ab
Step 4/7 : COPY --chown=tt-operator:tt-operator timestenclassic-operator
/usr/local/bin/timestenclassic-operator
 ---> Using cache
 ---> 6ccad53230f0
Step 5/7 : COPY --chown=tt-operator:tt-operator $TT_DISTRO /tt-operator/
$TT_DISTRO
 ---> 5cd31705485a
Step 6/7 : USER tt-operator
 ---> Running in 6a773ddac5dd
Removing intermediate container 6a773ddac5dd
 ---> 875ee38ebc75
Step 7/7 : ENTRYPOINT ["/usr/local/bin/timestenclassic-operator"]
 ---> Running in fed0f6c94c2f
Removing intermediate container fed0f6c94c2f
 ---> 10dde79e1617
Successfully built 10dde79e1617
Successfully tagged ttclassic-operator:3
```

3. Use the docker command to tag the Operator image.

   - Replace phx.ocir.io/youraccount with the location of your image registry. (phx.ocir.io/youraccount is represented in **bold** in this example.)
   - Replace ttclassic-operator:3 with the name you chose in the previous step. (ttclassic-operator is represented in **bold** in this example.)

```
% docker tag ttclassic-operator:3 phx.ocir.io/youraccount/ttclassic-operator:3
```

4. Use the docker command to push the Operator image to your registry.

   - Replace phx.ocir.io/youraccount with the location of your image registry. (phx.ocir.io/youraccount is represented in **bold** in this example.)
   - Replace ttclassic-operator:3 with the name you chose in the previous steps. (ttclassic-operator:3 is represented in **bold** in this example.)

```
% docker push phx.ocir.io/youraccount/ttclassic-operator:3
The push refers to repository [phx.ocir.io/youraccount/ttclassic-operator]
```

```
46458e9fc890: Pushed
471a399f0540: Pushed
9e51a2b82af3: Pushed
2f915858a916: Layer already exists
3: digest:
sha256:9b941f12e3d52298b9b38f7766ddcdfb1d011857a990ff01a8adafd32f3d3e8d size:
1166
```

You successfully built the Operator image and pushed it to your image registry.

Deploy the Operator

To deploy the Operator, you first customize it for your namespace and then deploy it. As a final step, you can verify the Operator is running. See "Deploying the Operator" on page 2-6 for information.

To customize the Operator for your namespace, navigate to the *kube_files*/deploy directory, and edit the `operator.yaml` file. This file is provided in the distribution that you previously unpacked. See "Downloading TimesTen and the TimesTen Operator" on page 2-2 for details.

1. Modify these fields represented in **bold** (in the `operator.yaml` file below):

   - `replicas: 1`

     Replace `1` with the number of copies of the Operator that you would like to run. `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 Kubernetes uses to pull images from your registry.
   - Replace `phx.ocir.io/youraccount` with the location of your image registry.
   - Replace `ttclassic-operator:3` with the name you chose in the previous steps.

```
% cd kube_files/deploy
% vi operator.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: timestenclassic-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: timestenclassic-operator
  template:
    metadata:
      labels:
        name: timestenclassic-operator
    spec:
      serviceAccountName: timestenclassic-operator
      imagePullSecrets:
      - name: sekret
      containers:
        - name: timestenclassic-operator
          image: phx.ocir.io/youraccount/ttclassic-operator:3
          command:
          - timestenclassic-operator
```

```
imagePullPolicy: Always
env:
  - name: WATCH_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
  - name: POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: OPERATOR_NAME
    value: "timestenclassic-operator"
  - name: GODEBUG
    value: "x509ignoreCN=0"
```

2. Use the `kubectl create` command to define the Operator to your namespace and to start the Operator.

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

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

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

```
% kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
timestenclassic-operator-846cb5c97c-9gv46   1/1     Running   0          37s
```

   You verified that the Operator is running.

Build the TimesTen image

Before you can start TimesTen in your Kubernetes cluster, you must first package TimesTen as a container image and then push the image to your image registry. The files that you need to do this are provided in the *kube_files* directory tree. See "Building the TimesTen image" on page 2-8 for information.

To run the TimesTen containers as non-root, you must uncomment `USER 333` in the *kube_files*`/ttimage/Dockerfile`.

To create the TimesTen container image, perform these steps:

1. Navigate to the *kube_files*`/ttimage` directory, and copy the TimesTen distribution into it. This example assumes you downloaded the `timesten1814110.server.linux8664.zip` distribution into the *installation_dir* directory. See "Download the TimesTen Operator" on page C-2 for information. Then, verify the `timesten1814110.server.linux8664.zip` file is in the *kube_files*`/ttimage` directory.

```
% cd kube_files/ttimage
% cp installation_dir/timesten1814110.server.linux8664.zip .
% ls *.zip
timesten1814110.server.linux8664.zip
```

2. Navigate to the *kube_files*`/ttimage` directory (if not already in this directory). Edit the `Dockerfile`, replacing `timesten1814110.server.linux8664.zip` with the name of your TimesTen full distribution. If your TimesTen distribution is `timesten1814110.server.linux8664.zip`, no modification is necessary. If not, the

modification you need to make is represented in **bold**. Note: The TimesTen full distribution must be `18.1.4.11.0` or later.

In addition, uncomment `USER 333` in this `Dockerfile` (represented in **bold**, in this example).

```
% cd kube_files/ttimage
% vi Dockerfile

# Copyright (c) 2019, 2021, Oracle and/or its affiliates.

FROM container-registry.oracle.com/os/oraclelinux:7

ARG TT_DISTRO=timesten1814110.server.linux8664.zip

RUN yum -y install tar gzip vim curl unzip libaio util-linux
RUN groupadd -g 333 oracle
RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle oracle
RUN install -d -m 0750 -o oracle -g oracle /home/oracle
COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop .ttdotversion agent2
create1.sql create2.sql get1.sql repcreate.sql repduplicate.sql runsql.sql
pausecg.sql /home/oracle/
# Uncomment the following line if you are using the optional non-root
installation procedure.
USER 333
ENTRYPOINT "/home/oracle/starthost.pl"
```

3. Use the `docker` command to build the TimesTen container image. Replace `tt1814110` with a name of your choosing (represented in **bold**, in the `docker build` command below). Note that the output may change from release to release.

```
% docker build -t tt1814110:3 .

Sending build context to Docker daemon  445.8MB
Step 1/9 : FROM container-registry.oracle.com/os/oraclelinux:7
 ---> d788eca028a0
Step 2/9 : ARG TT_DISTRO=timesten1814110.server.linux8664.zip
 ---> Using cache
 ---> a259a93fe906
Step 3/9 : RUN yum -y install tar gzip vim curl unzip libaio util-linux
 ---> Using cache
 ---> ac676b5376f3
Step 4/9 : RUN groupadd -g 333 oracle
 ---> Using cache
 ---> ce16920f085c
Step 5/9 : RUN useradd -M -d /tt/home/oracle -s /bin/bash -u 333 -g oracle
oracle
 ---> Using cache
 ---> 0319814aca1c
Step 6/9 : RUN install -d -m 0750 -o oracle -g oracle /home/oracle
 ---> Using cache
 ---> c8612b53398a
Step 7/9 : COPY --chown=oracle:oracle $TT_DISTRO /home/oracle/
 ---> 31cae98b71fd
Step 8/9 : COPY --chown=oracle:oracle .bashrc starthost.pl .ttdrop
.ttdotversion agent2 create1.sql create2.sql get1.sql repcreate.sql
repduplicate.sql runsql.sql pausecg.sql /home/oracle/
 ---> e50eb99c9b54
Step 9/9 : ENTRYPOINT "/home/oracle/starthost.pl"
 ---> Running in 0b41efd38837
```

```
Removing intermediate container 0b41efd38837
 ---> 171245e546d5
Successfully built 171245e546d5
Successfully tagged tt1814110:3
```

4. Use the `docker` command to tag the TimesTen container image. Replace the following, represented in **bold**, in the `docker tag` command below.
   - `tt1814110:3` with the name you chose in the previous step.
   - `phx.ocir.io/youraccount` with the location of your image registry.

   ```
   % docker tag tt1814110:3 phx.ocir.io/youraccount/tt1814110:3
   ```

5. Use the `docker` command to push the TimesTen container image to your registry. Replace the following, represented in **bold**, in the `docker push` command below.
   - `phx.ocir.io/youraccount` with the location of your image registry.
   - `tt1814110:3` with the name you chose previously.

   ```
   % docker push phx.ocir.io/youraccount/tt1814110:3

   The push refers to repository [phx.ocir.io/youraccount/tt1814110]
   97a0f250b2fe: Pushed
   650b003a3ad4: Pushed
   b8de51528854: Pushed
   62192d26e325: Pushed
   7dfe13e9b5a4: Pushed
   d8570fce965c: Pushed
   2f915858a916: Layer already exists
   3: digest:
   sha256:a6ac313394229eb2256d4a56fbcd8e2eda50ea2cc21991fa76f11701f2299710
   size: 1788
   ```

You successfully built the TimesTen container image. It is pushed to your image registry.

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 "Understanding the configuration metadata and the Kubernetes facilities" on page 3-1 for information on the configuration files and the ConfigMap facility.

On your Linux development host, perform these steps:

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 `scott` user with the `tiger` password.

```
vi adminUser

scott/tiger
```

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 `scott` user. The Operator will automatically initialize your database with these object definitions.

```
vi schema.sql

create sequence scott.s;
create table scott.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:
----
scott/tiger

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

```
schema.sql:
----
create sequence scott.s;
create table scott.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" on page 4-2 and "The TimesTenClassic object type" on page 11-1 for detailed information on the TimesTenClassic object.

To run the TimesTen containers as non-root, you must add `SecurityContext` as illustrated below.

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" on page 11-3 for information on the fields that you must specify in this YAML file as well as the fields that are optional.

   To run the TimesTen containers as non-root, add `.template.spec.securityContext` information (represented in **bold**, in this example).

   In addition, 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 `250G` 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 `250G` for `storageSize`. For demonstration purposes, a value of `50G` is adequate. See the `storageSize` and the `logStorageSize` entries in the Table 11–3, " TimesTenClassicSpecSpec" for information.
   - `image`: Replace `phx.ocir.io/youraccount/tt1814110:3` with the location of the image registry (`phx.ocir.io/youraccount`) and the image containing TimesTen (`tt1814110:3`)
   - `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" on

page 3-6 and "Example using one ConfigMap" on page 3-7 for information on ConfigMaps.

```
% vi sample.yaml

apiVersion: timesten.oracle.com/v1
kind: TimesTenClassic
metadata:
  name: sample
spec:
  ttspec:
    storageClassName: oci
    storageSize: 250G
    image: phx.ocir.io/youraccount/tt1814110:3
    dbConfigMap:
    - sample
    imagePullSecret: sekret
 template:
    spec:
      securityContext:
        fsGroup: 333
      containers:
      - name: tt
        securityContext:
          runAsUser: 333
          runAsGroup: 333
```

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 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   3m33s
```

3. Use the `kubectl describe` command again to view the active standby pair provisioning in detail. Note the `Security Context` (represented in **bold**) is included in the active standby pair provisioning, indicating the TimesTen containers are running as non-root.

```
% kubectl describe ttc sample
Name:         sample
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>
API Version:  timesten.oracle.com/v1
Kind:         TimesTenClassic
Metadata:
  Creation Timestamp:  2021-04-09T22:01:34Z
  Generation:          1
  Resource Version:    148466657
  Self Link:
   /apis/timesten.oracle.com/v1/namespaces/mynamespace/timestenclassics/sample
  UID:                 25ff6f40-997f-11eb-86a3-06b2b9dd76bc
Spec:
  Template:
    Spec:
      Containers:
        Name:  tt
        Security Context:
          Run As Group:  333
          Run As User:   333
      Security Context:
        Fs Group:  333
  Ttspec:
    Db Config Map:
      sample
    Image:               phx.ocir.io/youraccount/tt1814110:3
    Image Pull Policy:   Always
    Image Pull Secret:   sekret
    Storage Class Name:  oci
    Storage Size:        250G
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-0
  High Level State:              Normal
```

```
Last Event:                    27
Last High Level State Switch:  1618005884
Pod Status:
  Cache Status:
    Cache Agent:        Not Running
    Cache UID Pwd Set:  true
    N Cache Groups:     0
  Db Status:
    Db:                          Loaded
    Db Id:                       34
    Db Updatable:                Yes
  Initialized:                   true
  Last High Level State Switch:  ?
  Pod Status:
    Agent:                Up
    Last Time Reachable:  1618006024
    Pod IP:               10.244.7.37
    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:                    ACTIVE
  Times Ten Status:
    Daemon:          Up
    Instance:        Exists
    Release:         18.1.4.11.0
  Admin User File:   true
  Cache User File:   false
  Cg File:           false
  Disable Return:    false
  High Level State:  Healthy
  Intended State:    Active
  Local Commit:      false
  Name:              sample-0
  Schema File:       true
  Using Twosafe:     false
  Cache Status:
    Cache Agent:        Not Running
    Cache UID Pwd Set:  true
    N Cache Groups:     0
  Db Status:
    Db:                          Loaded
    Db Id:                       34
    Db Updatable:                No
  Initialized:                   true
  Last High Level State Switch:  ?
  Pod Status:
    Agent:                Up
    Last Time Reachable:  1618006024
    Pod IP:               10.244.14.33
    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:            18.1.4.11.0
    Admin User File:      true
    Cache User File:      false
    Cg File:              false
    Disable Return:       false
    High Level State:     Healthy
    Intended State:       Standby
    Local Commit:         false
    Name:                 sample-1
    Schema File:          true
    Using Twosafe:        false
  Prev High Level State:  Initializing
  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
  Status Version:         1.0
Events:
  Type  Reason       Age    From       Message
  ----  ------       ----   ----       -------
  -     Create       5m34s  ttclassic  Service sample created
  -     Create       5m34s  ttclassic  StatefulSet sample created
  -     Create       5m34s  ttclassic  Secret
tt25ff6f40-997f-11eb-86a3-06b2b9dd76bc created
  -     StateChange  4m12s  ttclassic  Pod sample-1 Daemon Up
  -     StateChange  4m12s  ttclassic  Pod sample-0 Agent Up
  -     StateChange  4m12s  ttclassic  Pod sample-0 Release 18.1.4.11.0
  -     StateChange  4m12s  ttclassic  Pod sample-1 Agent Up
  -     StateChange  4m12s  ttclassic  Pod sample-1 Release 18.1.4.11.0
  -     StateChange  4m12s  ttclassic  Pod sample-0 Daemon Up
  -     StateChange  3m2s   ttclassic  Pod sample-0 Database Loaded
  -     StateChange  3m2s   ttclassic  Pod sample-0 Database Updatable
  -     StateChange  3m2s   ttclassic  Pod sample-0 CacheAgent Not Running
  -     StateChange  3m2s   ttclassic  Pod sample-0 RepAgent Not Running
  -     StateChange  3m2s   ttclassic  Pod sample-0 RepState IDLE
  -     StateChange  3m2s   ttclassic  Pod sample-0 RepScheme None
  -     StateChange  3m1s   ttclassic  Pod sample-0 RepAgent Running
  -     StateChange  3m1s   ttclassic  Pod sample-0 RepScheme Exists
  -     StateChange  3m1s   ttclassic  Pod sample-0 RepState ACTIVE
  -     StateChange  2m42s  ttclassic  Pod sample-1 Database Loaded
  -     StateChange  2m42s  ttclassic  Pod sample-1 Database Not Updatable
  -     StateChange  2m42s  ttclassic  Pod sample-1 CacheAgent Not Running
  -     StateChange  2m42s  ttclassic  Pod sample-1 RepAgent Not Running
  -     StateChange  2m42s  ttclassic  Pod sample-1 RepScheme Exists
  -     StateChange  2m42s  ttclassic  Pod sample-1 RepState IDLE
  -     StateChange  2m31s  ttclassic  Pod sample-1 RepAgent Running
  -     StateChange  2m31s  ttclassic  Pod sample-1 RepState STANDBY
  -     StateChange  2m25s  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 TimesTen container runs as non-root

You can run the `kubectl exec` command to invoke shells in your Pods and control TimesTen, which is running in those Pods. Run the Linux `id` command to verify the UID is `333`.

1. Establish a shell in the Pod and run the Linux `id` command.

```
% kubectl exec -it sample-0 -c tt -- /usr/bin/bash
% id
uid=333(oracle) gid=333(oracle) groups=333(oracle)
```

2. Verify you can connect to the `sample` database and run a simple query.

```
% ttisql sample
Copyright (c) 1996, 2020, 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=oracle;DataStore=/tt/home/oracle/datastore/sample;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=US7ASCII;AutoCreate=0;
PermSize=200;DDLReplicationLevel=3;ForceDisconnectEnabled=1;
(Default setting AutoCommit=1)

Command> SELECT * FROM dual;
< X >
1 row found.
```

Index

A

active standby pair deployment
  monitor, 4-3
ActiveDown
  state of active standby pair, 6-3
ActiveTakeover
  health of active standby pair, 6-3
adminUser
  metadata file, 3-2

B

BothDown
  handling, 6-5
  state of active standby pair, 6-4

C

cachegroups.sql
  metadata file, 3-2
cacheUser
  metadata file, 3-3
CatchingUp
  health of Pod, 6-2
client/server drivers
  using, 5-3
complete example
  build Operator image, A-4
  build TimesTen image, A-7, C-7
  cleanup, A-19
  configure Kubernetes, A-3
  create ConfigMap, A-9
  create TimesTenClassic object, A-10
  deploy Operator, A-5
  deploy TimesTenClassic CRD, A-4
  download TimesTen Operator, A-2
  monitor deployment, A-12
  recover from failure, A-18
  set up environment, A-1
  verify connection to active TimesTen database, A-17
  verify underlying objects, A-16
concepts
  Kubernetes, 1-1
ConfigMaps
  using, 3-6
configuration metadata, 3-1
configuration options
  persistent storage, 3-13
ConfiguringActive
  state of active standby pair, 6-4
connection to database
  verify, 4-8
containers
  Kubernetes, 1-1
CRD
  definition, 1-2
csWallet
  metadata file, 3-4
Customer Resource Definition
  definition, 1-2

D

db.ini
  metadata file, 3-4
Deployment
  definition, 1-1
deployment
  TimesTen databases, 4-1
  TimesTenClassic CRD, 2-4
direct mode applications
  using, 5-1
Down
  health of Pod, 6-2

E

encryption
  client for TimesTen client/server,configure client-side attributes, 8-18
  client for TimesTen client/server,copy client wallet, 8-18
  configuration on TimesTen Server,metadata files, 8-11
  configure client for TimesTen client/server, 8-17
  configure on TimesTen server,create Kubernetes facilities, 8-12
  configure TLS for client/server, 8-11
  configure TLS for replication, 8-3
  configure TLS on TimesTen server, 8-11

create ConfigMap, 8-5
create Kubernetes facilities, 8-3
create Kubernetes Secret, 8-4
create metadata files, 8-3
create TimesTenClassic object, 8-6
create TLS certificates, 8-1
monitor deployment of TimesTenClassic object, 8-8
TimesTen server,create ConfigMap, 8-13
TimesTen server,create TimesTenClassic object, 8-15
TimesTen server,monitor deployment, 8-16
verify TLS for replication, 8-9
epilog.sql
metadata file, 3-5
example
failover and recovery process, 9-1
using one ConfigMap, 3-7
using one ConfigMap and one Secret, 3-9

F

Failed
state of active standby pair, 6-4
failover
handling, 9-1
failover and recovery process
example, 9-1

H

Handling BothDown, 6-5
health of active standby pair
ActiveTakeover, 6-3
health of active standby pair of databases
tools to monitor, 6-3
health of each pod
monitoring, 6-1
health of Pod
CatchingUp, 6-2
Down, 6-2
Healthy, 6-2
HealthyActive, 6-2
HealthyStandby, 6-2
OtherDown, 6-2
Terminal, 6-2
Unknown, 6-2
UpgradeFailed, 6-3
Healthy
health of Pod, 6-2
HealthyActive
health of Pod, 6-2
HealthyStandby
health of Pod, 6-2

I

init container
using, 3-12
Initializing
state of active standby pair, 6-4

K

Kubernetes
concepts, 1-1
containers, 1-1
Kubernetes facilities, 3-1
Kubernetes objects
definition, 1-4
Kubernetes Operator
definition, 1-2

M

managing
delete active standby pair of TimesTen databases, 6-29
manually invoke TimesTen utilities, 6-19
modify TimesTen connection attributes, 6-19
revert to manual control, 6-26
ManualInterventionRequired
state of active standby pair, 6-4
metadata file
adminUser, 3-2
cachegroups.sql, 3-2
cacheUser, 3-3
csWallet, 3-4
db.ini, 3-4
epilog.sql, 3-5
replicationWallet, 3-5
schema.sql, 3-5
sqlnet.ora, 3-5
tnsnames.ora, 3-6
modify
first connection attributes, 6-22
general connection attributes, 6-24
modify TimesTen connection attributes
manually edit db.ini file, 6-20
monitor
active standby deployment, 4-3

N

non-root installation
build the Operator image, C-4
configure Kubernetes, C-3
create ConfigMap, C-9
create TimesTenClassic object, C-11
deploy Operator, C-6
deploy TimesTenClassic CRD, C-4
download TimesTen Operator, C-2
monitor deployment, C-12
set up environment, C-1
verify TimesTen container, C-16
Normal
state of active standby pair, 6-4

O

Operator
definition, 1-4
tools to locate, 6-18

upgrade, 10-2
OtherDown
health of Pod, 6-2

P

persistent storage
configuration options, 3-13
PersistentVolume
definition, 1-1
Pod
definition, 1-1
Pod location, 3-15
prerequisites, 2-1

R

recovery
handling, 9-1
Reexamine
state of active standby pair, 6-4
replicationWallet
metadata file, 3-5
resource specification
tt and daemonlog containers, 3-14

S

schema.sql
metadata file, 3-5
Secrets
using, 3-6
Service
definition, 1-2
sqlnet.ora
metadata file, 3-5
standby database, 10-23
StandbyCatchup
state of active standby pair, 6-5
StandbyDown
state of active standby pair, 6-5
StandbyStarting
state of active standby pair, 6-5
state of active standby pair
ActiveDown, 6-3
BothDown, 6-4
ConfiguringActive, 6-4
Failed, 6-4
Initializing, 6-4
ManualInterventionRequired, 6-4
Normal, 6-4
Reexamine, 6-4
StandbyCatchup, 6-5
StandbyDown, 6-5
StandbyStarting, 6-5
WaitingforActive, 6-5
StatefulSet
definition, 1-1
supported metadata files, 3-1

T

Terminal
health of Pod, 6-2
TimesTen agent, 1-7
TimesTen Cache
cachegroups.sql metadata file, 7-1
cacheUser metadata file, 7-1
clean up cache metadata, 7-10
create Kubernetes facilities, 7-3
create metadata files, 7-3
create TimesTenClassic object, 7-6
db.ini metadata file, 7-2
monitor deployment of TimesTenClassic object, 7-7
overview, 7-1
schema.sql metadata file, 7-2
sqlnet.ora metadata file, 7-2
tnsnames.ora metadata file, 7-2
TimesTen Cache example
cleanup, B-19
create metadata files, B-6
create Oracle Database tables, B-4
create Oracle Database users, B-1
create TimesTenClassic object, B-11
grant privileges to cache administration user, B-3
monitor deployment, B-12
performing operations on cache group tables, B-16
set up Oracle database, B-1
verify TimesTen Cache configuration, B-15
TimesTen containers, 1-7
TimesTen databases
tools for managing, 6-18
TimesTen Operator
customize Operator, 2-6
deployment, 1-7
overview, 1-2
TimesTen Operator installation, 2-1
build Operator image, 2-5
build TimesTen image, 2-8
deploy Operator, 2-6
download requirements, 2-2
Kubernetes service account, 2-4
prerequisites, 2-1
verify Operator, 2-7
TimesTen upgrade, 10-10
TimesTenClassic
syntax, 11-2
TimesTenClassic CRD
deployment, 2-4
TimesTenClassic object
create, 4-2
define, 4-2
overview, 11-1
TimesTenClassic object type
definition, 1-3
description, 11-1
TimesTenClassic state
monitor, 4-3
TimesTenClassicSpec

syntax, 11-2
TimesTenClassicSpecSpec
syntax, 11-3
TimesTenClassicStatus
description, 11-8
tnsnames.ora
metadata file, 3-6
/ttconfig directory, 3-6

U

underlying objects
verify existence, 4-8
Unknown
health of Pod, 6-2
upgrade, 10-23
fail over, 10-27
Operator, 10-2
standby database, 10-23
TimesTen, 10-10
upgrade each TimesTenClassic object
modify TimesTenClassic object, 10-14, 10-21
upgrade Operator
build new Operator image, 10-5
download new release, 10-3
review current Operator, 10-6
upgrade process
overview, 10-1
upgrade TimesTen
build new TimesTen image, 10-10
UpgradeFailed
health of Pod, 6-3

W

WaitingforActive
state of active standby pair, 6-5