About Terracotta Ehcache

Innovation Release

Version 10.2

April 2018

TERRACOTTA

This document applies to Terracotta Ehcache Version 10.2 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: TC-EHC-AE-102-20180417

Table of Contents

1 Introduction to Terracotta Ehcache

High-Level Overview

Terracotta Ehcache is a standards-based cache that boosts performance, offloads your database, and simplifies scalability. It's the most widely-used Java-based cache because it's robust, proven, full-featured, and integrates with other popular libraries and frameworks. Terracotta Ehcache scales from in-process caching, all the way to mixed in-process/out-of-process deployments with terabyte-sized caches.

Terracotta Ehcache strengthens the distributed caching capabilities via a new generation of the Terracotta Server with support for high-availability and improved performance.

Features include:

- An API that leverages Java generics and simplifies Cache interactions,
- Full compatibility with javax.cache API (JSR-107),
- Offheap storage capabilities, including offheap-only caches,
- Out of the box Spring Caching and Hibernate integration thanks to the javax.cache support,
- And many more ...

Terracotta Ehcache is the commercial release of Ehcache. Ehcache also is alive as an open source project. Builds are available on the project's GitHub release page at "https://github.com/ehcache/ehcache3/releases".

Product Components

Terracotta Ehcache consists of the following components:

Component	Description
Ehcache API	The Ehcache API.
Terracotta Server	The server component that manages the resources of all of the Ehcache caches.
Terracotta Management Console	The browser-based graphical user interface that allows administrators to monitor and manage the current status of all Ehcache caches managed by the Terracotta Server. If there are multiple Terracotta Servers in use in a cluster, these can also be monitored and managed.

If you wish to use the product purely as a Java programmer, you require the components Ehcache API and Terracotta Server.

If you wish to perform administration of all runtime components and the Terracotta Server using a graphical user interface, you require the Terracotta Management Console in addition to the other components.

Installation Procedures

For instructions on how to install the product, refer to the document *Terracotta Ehcache Installation Guide*

2 What is Ehcache?

Features

Ehcache is the most widely-used Java-based cache. It is:

- robust,
- proven,
- full-featured,
- and integrates with other popular libraries and frameworks.

Ehcache scales from in-process caching, all the way to mixed in-process/out-of-process deployments with terabyte-sized caches.

Fast and Lightweight

Fast

Ehcache's concurrency features are designed for large, high concurrency systems.

Simple

Many users of Ehcache hardly know they are using it. Sensible defaults require no initial configuration.

Small footprint

Ehcache strives to maintain a small footprint - keeping your apps as light as they can be.

Minimal dependencies

The only dependency for core use is SLF4J.

Scalable

Provides for scalability into terabytes

The largest Ehcache installations utilize multiple terabytes of data storage.

With off-heap storage, Ehcache has been tested to store 6TB of data in a single process (JVM).

Scalable to hundreds of caches

The largest Ehcache installations use hundreds of caches.

Tuned for high concurrent load on large/wide multi-CPU servers

Ehcache is specifically built and tested to run well under highly concurrent access on systems with dozens of cores. This results in an optimal balance between thread safety and performance.

Flexible

Provides multiple strategies for:

- Expiration policies
- Storage tiers (on-heap, off-heap, disk, clustered)
- Configuration of caches

Standards Based

Support of JSR-107 JCache - Java Temporary Caching API

You can use Ehcache as a JCache provider. This allows you to use JCache API calls to develop a complete application, without the need to use any Ehcache API calls.

Distributed Caching

Ehcache supports simple yet high performance distributed caching.

Enterprise Java and Applied Caching

High quality implementations for common caching scenarios and patterns.

Cacheable Commands

This is the trusty old command pattern with a twist: asynchronous behavior, fault tolerance and caching. Creates a command, caches it and then attempts to execute it.

Works with Hibernate

Ehcache is popularly used as Hibernate's second-level cache.

Transactional support through JTA

Ehcache supports JTA and is a fully XA compliant resource participating in the transaction, two-phase commit and recovery.

See the complete Transaction Module Java Documentation at "www.ehcache.org".

API Documentation

The Javadoc documentation of the API can be found here:

"www.ehcache.org/documentation/"

Open Source Kit

There is an open-source version of Ehcache, paired with Terracotta Server open source functionality. This can be found at "http://www.terracotta.org/open-source/".

Basic Terms

Cache

A cache is a collection of temporary data that either duplicates data located elsewhere or is the result of a computation. Data that is already in the cache can be repeatedly accessed with minimal costs in terms of time and resources.

Cache Entry

A cache entry consists of a key and its mapped data value within the cache.

Cache hit

When a data element is requested from cache and the element exists for the given key, it is referred to as a *cache hit* (or simply, "a hit").

Cache miss

When a data element is requested from cache and the element does not exist for the given key, it is referred to as a *cache miss* (or simply, "a miss").

Eviction

When entries are removed from the cache in order to make room for newer entries (typically when the cache has run out of data storage capacity), it is referred to as eviction.

Expiration

When entries are removed from the cache after some defined amount of time has passed, it is referred to as expiration.

Hot Data

Data that has recently been used by an application is very likely to be accessed again soon. Such data is considered *hot*. A cache may attempt to keep the hottest data most quickly available, while attempting to choose the least hot data for eviction.

System-of-Record

The system-of-record is the authoritative source of truth for the data. The cache acts as a local copy of data retrieved from or stored to the system-of-record (SOR). The SOR is often a traditional database, although it might be a specialized file system or some other reliable long-term storage. It can also be a conceptual component such as an expensive computation.

Data Freshness and Expiration

Data Freshness

Data *freshness* describes how up-to-date a copy of data (e.g. in a cache) is compared to the source version of the data (e.g. in the system-of-record (SoR). A *stale* copy is considered to be out of sync (or likely to be out of sync) with the SoR.

Databases (and other SORs) weren't built with caching outside of the database in mind, and therefore don't normally come with any default mechanism for notifying external processes when data has been updated or modified. Thus external components that have loaded data from the SoR have no direct way of ensuring that data is not stale.

Cache Entry Expiration

Ehcache can assist you with reducing the likelihood that stale data is used by your application by *expiring* cache entries after some amount of configured time. Once expired, the entry is automatically removed from the cache.

For instance, the cache could be configured to expire entries five seconds after they are put into the cache - which is a time-to-live *TTL* setting. Or to expire entries 17 seconds after the last time the entry was retrieved from the cache - which is a time-to-idle *TTI* setting.

Note: TTI is not supported for caches with clustered storage tiers.

The expiration configuration that would be most appropriate for your cache (if any) would be a mixture of a business and technical decision based upon the requirements and assumptions of your application.

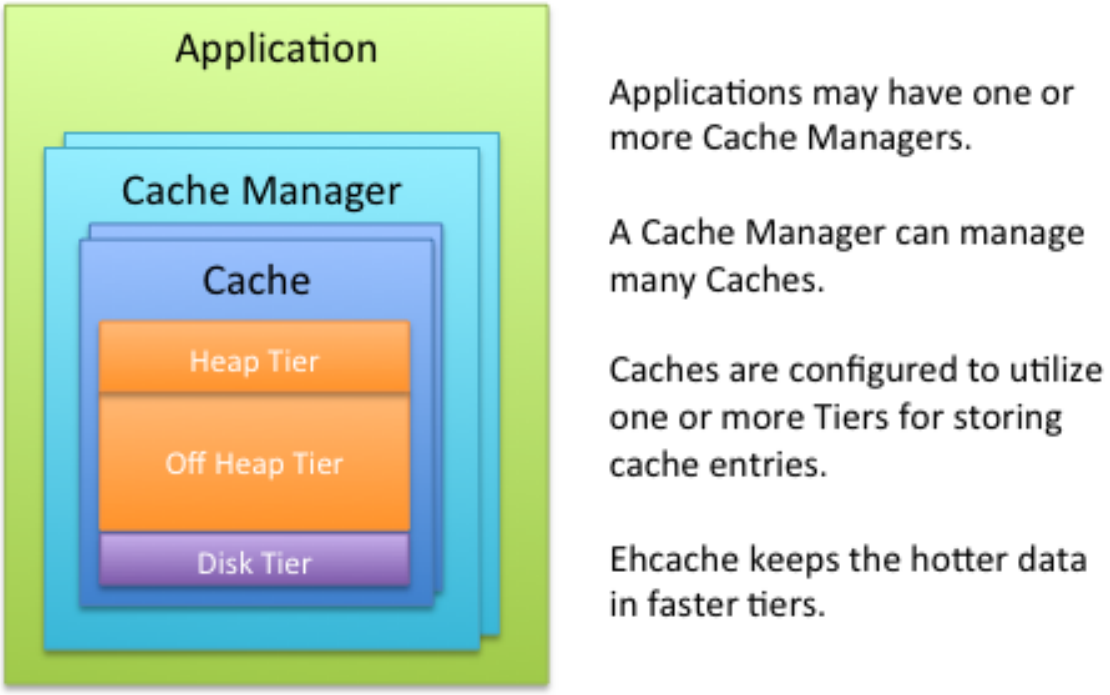
Storage Tiers

You can configure Ehcache to use various data storage areas. When a cache is configured to use more than one storage area, those areas are arranged and managed as *tiers*. They are organized in a hierarchy, with the lowest tier being called the *authority* tier and the others being part of the *caching* tier. The caching tier can itself be composed of more than one storage area. The *hottest* data is kept in the caching tier, which is typically less abundant but faster than the authority tier. All the data is kept in the authority tier, which is slower but more abundant.

Data stores supported by Ehcache include:

- **On-Heap Store** – Utilizes Java's on-heap RAM memory to store cache entries. This tier utilizes the same heap memory as your Java application, all of which must be scanned by the JVM garbage collector. The more heap space your JVM utilizes the more your application performance will be impacted by garbage collection pauses. This store is extremely fast, but is typically your most limited storage resource.

- **Off-Heap Store** – Limited in size only by available RAM. Not subject to Java garbage collection (GC). Is quite fast, yet slower than the On-Heap Store because data must be moved to and from the JVM heap as it is stored and re-accessed.
- **Disk Store** – Utilizes a disk (file system) to store cache entries. This type of storage resource is typically very abundant but much slower than the RAM-based stores. As for all applications using disk storage, it is recommended to use a fast and dedicated disk to optimize the throughput.
- **Clustered Storage** – This data store is a cache on a remote server. The remote server may optionally have a failover server providing improved high availability. Since clustered storage comes with performance penalties due to such factors as network latency as well as for establishing client/server consistency, this tier, by nature, is slower than local off-heap storage.



Topology Types

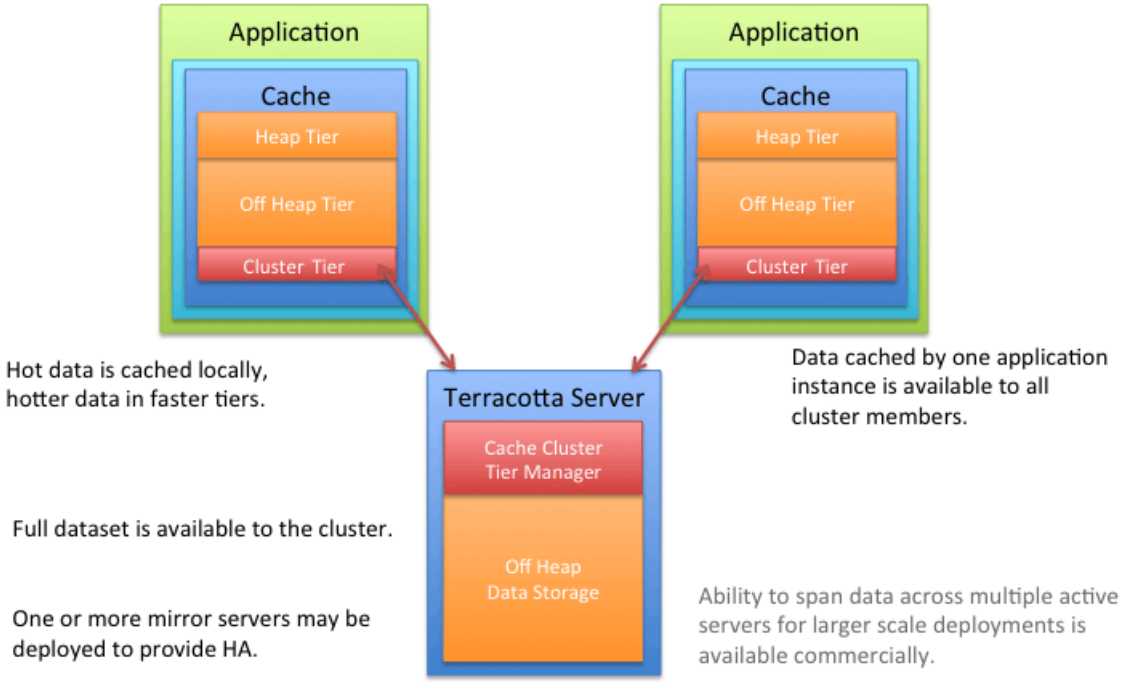
The following describes the basic types of caching topologies:

Standalone

The data set is held in the application node. Any other application nodes are independent with no communication between them. If a standalone topology is used where there are multiple application nodes running the same application, then their caches are completely independent.

Distributed / Clustered

The data is held in a remote server (or array of servers) with a subset of hot data held in each application node. This topology offers a selection of consistency options. A distributed topology is the recommended approach in a clustered or scaled-out application environment. It provides the best combination of performance, availability, and scalability.



It is common for many production applications to be deployed in clusters of multiple instances for availability and scalability. However, without a distributed cache, application clusters exhibit a number of undesirable behaviors, such as:

- **Cache Drift** - If each application instance maintains its own cache, updates made to one cache will not appear in the other instances. A distributed cache ensures that all of the cache instances are kept in sync with each other.
- **Database Bottlenecks** - In a single-instance application, a cache effectively shields a database from the overhead of redundant queries. However, in a distributed application environment, each instance must load and keep its own cache fresh. The overhead of loading and refreshing multiple caches leads to database bottlenecks as more application instances are added. A distributed cache eliminates the per-instance overhead of loading and refreshing multiple caches from a database.

3 What is the Terracotta Server?

The Terracotta Server provides the distributed data platform for Terracotta products. A cluster of Terracotta Servers configured to work together is referred to as a Terracotta Server Array (TSA). A Terracotta Server Array can vary from a single server, to a basic two-server tandem for High Availability (HA), to a multi-server array providing configurable scale, high performance, and deep failover coverage.

The main features of the Terracotta Server include:

- **Distributed In-memory Data Management**

  Manages 10-100x more data in memory than data grids

- **Scalability Without Complexity**

  Simple configuration and deployment option for scaling-up and/or scale-out to meet growing demand and facilitate capacity planning

- **High Availability**

  Instant failover for continuous uptime and services

- **Configurable Health Monitoring**

  Terracotta health checker monitors client and server health

- **Persistent Application State**

  Automatic permanent storage of all current shared in-memory data with ultra-fast recovery upon server restarts

- **Automatic Node Reconnection**

  Temporarily disconnected server instances and clients rejoin the cluster without operator intervention

4 What is the Terracotta Management Console?

The Terracotta Management Console (TMC) is a web-based administration and monitoring application with many capabilities and advantages, including the following:

- Feature-rich and easy-to-use interface
- Remote management capabilities requiring only a web browser and network connection
- Visualize cluster topologies, monitor health, and manage Terracotta Servers
- Aggregates performance and usage statistics from multiple Terracotta nodes
- Cross-platform deployment
- Flexible deployment model, which can plug into both development environments and secure production architectures