Terracotta Management and Monitoring

Version 10.15

October 2022

TERRACOTTA

This document applies to Terracotta 10.15 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-TMC-UG-1015-20221101

Table of Contents

About This Documentation

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at https://knowledge.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://hub.docker.com/u/softwareag and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Getting Started with the Terracotta Management Console

About the Terracotta Management Console

This document describes the Terracotta Management Console (TMC), which is a browser-based application served by the Terracotta Management Server (TMS). The TMC provides a complete view of your Terracotta Server Array (TSA) and connected clients.

With the TMC you can:

- observe the cluster topology and categories of connected clients
- view statistics
- clear cache contents
- and more

Installing the TMS

The TMS is a standard Java web application and bundled web container. It gathers information from throughout the cluster and serves it to the TMC for display in your browser. The TMS is located in the installed Terracotta kit under the `tools/management` directory.

Configuring the TMS

Certain aspects of the TMS can be customized via the properties file located in the installed Terracotta kit at `tools/management/conf/tmc.properties`. While that file contains many more properties, those that follow are the most likely to be useful to the TMS administrator:

```
server.port=9480
tms.defaultUrl=terracotta://localhost:9410
spring.pid.file=${tmc.home:build/tmc}/run/tmc.pid
tms.storageFolder=target/tmc/data
tms.offheapSizeMB=1024
tms.persistenceMode=HYBRID
tms.offheapMonitoringFrequencyMs=30000
tms.offheapThreshold=80
tms.statisticsMaxAgeMinutes=1500
tms.statisticsScavengerFrequencyMs=30000
tms.eventLogMaxRecords=5000
```

```
tms.eventLogScavengerFrequencyMs=60000
```

Security

Please see the section *TMS Security* in the *Terracotta Server Administration Guide* for information on how to configure TMS security.

In essence, you need to configure the following properties in the `tmc.properties` file:

```
tms.security.root.directory
tms.security.audit.directory
tms.security.https.enabled
tms.security.authentication.scheme
tms.security.authorization.scheme
tms.security.root.directory.connection.default
```

and provide the correct files in the security root directory.

Starting and Stopping the TMS

To start the TMS, execute the script `start.bat` (on Windows) or `start.sh` (on UNIX-based systems), located in `tools/management/bin` under the installed Terracotta kit.

To terminate the TMS, use the associated `stop.bat` or `stop.sh` script, or platform-provided process management tools.

Adding Manageability to your Ehcache CacheManager

In order to get the fullest manageability and monitorability you must configure your `CacheManager` to make use of a `ManagementRegistryService`:

```
import org.ehcache.config.builders.CacheManagerBuilder;
import org.ehcache.management.registry.DefaultManagementRegistryConfiguration;
CacheManager cacheManager = CacheManagerBuilder.newCacheManagerBuilder()
    .using(new DefaultManagementRegistryConfiguration()
        .setCacheManagerAlias("MyCacheManager")
        .addTags("my-client-tag", "another-client-tag"))
    .build(true);
```

Configuring the management registry will activate it and will allow various capabilities to be exposed via the TMS REST interface with the alias and tags you provide. We recommend you to assign the same alias to equivalent instances of your CacheManager, across different clients so as to support statistics aggregation.

If you *do not* make use of a management registry, your CacheManager will still appear in the TMC but most management and monitoring features will not operate.

Tip:
If you want to exclude a cache manager from the TMC UI, you can add the following tag: `tmc-excluded`.

For related information, see the section *Ehcache API Developer Guide > Management and Monitoring with Ehcache*.

Adding Manageability to your TCStore Dataset

Unlike for an Ehcache `CacheManager`, there is no need to explicitly add manageability to your TCStore `Dataset`.

Still, if you are using a clustered dataset manager, it is good practice to set an alias to your dataset and some tags:

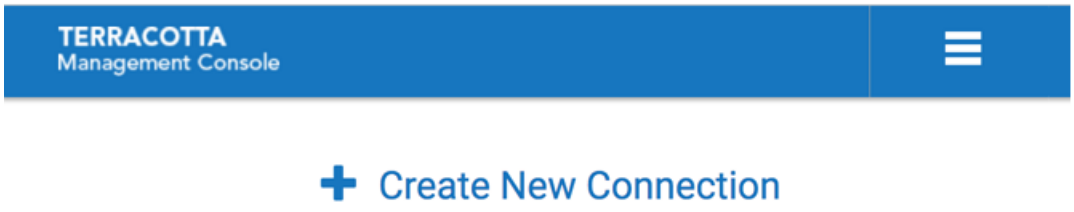
```
DatasetManager datasetManager = DatasetManager.clustered([...])
    .withClientAlias("myDsManager")
    .withClientTags("node-1", "webapp-2", "testing")
    .build();
```

Tip:
If you want to exclude a dataset manager from the TMC UI, you can add the following tag: `tmc-excluded`.

Connecting to the TMC

After starting the TMS, open a browser and visit *http://localhost:9480*. The TMC will load and present you with the Home Page, where persistent connections to your clusters can be created, viewed and managed.

Screenshot: TMC Home Page, Create New Connection



2 Using the Terracotta Management Console

The TMC Home Page

The TMC home page is where you:

- create/delete persistent connections to your cluster(s)
- optionally modify your connection properties
- view
  - the status of the servers that make up your cluster
  - the various categories of clients making use of your cluster
  - the server entities that are contained by the cluster and to which clients connect
- drill-down/jump to various presentations such as statistics and monitoring relating to those servers, clients, and entities
- take actions, such a clearing the contents of a cache

Tip:
TMC requires browser cookies to work. Please make sure your browser doesn't block reading cookies (some browser plugins can block reading cookies). If you see a related error message, you can try another browser, or use incognito mode, or disable browser plugins.

For information about starting and stopping Terracotta servers, refer to the section *Starting and Stopping the Terracotta Server* in the *Terracotta Server Administration Guide*.

For information about creating Terracotta clusters, refer to the section *The Cluster Tool* in the *Terracotta Server Administration Guide*.

Connections and Global Settings

To create a persistent connection to a particular operational cluster:

1. Click *Create New Connection*
2. In the *Connection URL* input area enter a URL addressing at least a single, running member of the fully configured Terracotta Server Array (TSA):

Screenshot: TMC Home Page. Connection URL to at least one running member of the TSA

Terracotta Server URL

```
terracotta://<server-host>:<listen-port>[,<server-host>:<listen-port>]*
```

If the TMS is able to connect to the specified server, it will request the complete TSA topology, persisting the addresses of each server in its database. This means that, in the future, the TMS will be able to connect to the TSA, even if the originally specified server should be unreachable, as long as at least one member of the TSA is running.

If you attempt to connect to a running server that is not part of a *configured* cluster, the TMS will return an error indicating that no license was discovered.

Note:

It is a best-practice to specify the addresses of all the members of a particular TSA stripe in the connection URL to support high-availability. Simply comma-separate each server's address (host:port).

The TMS will attempt to connect to each server, in turn, until a successful connection is established.

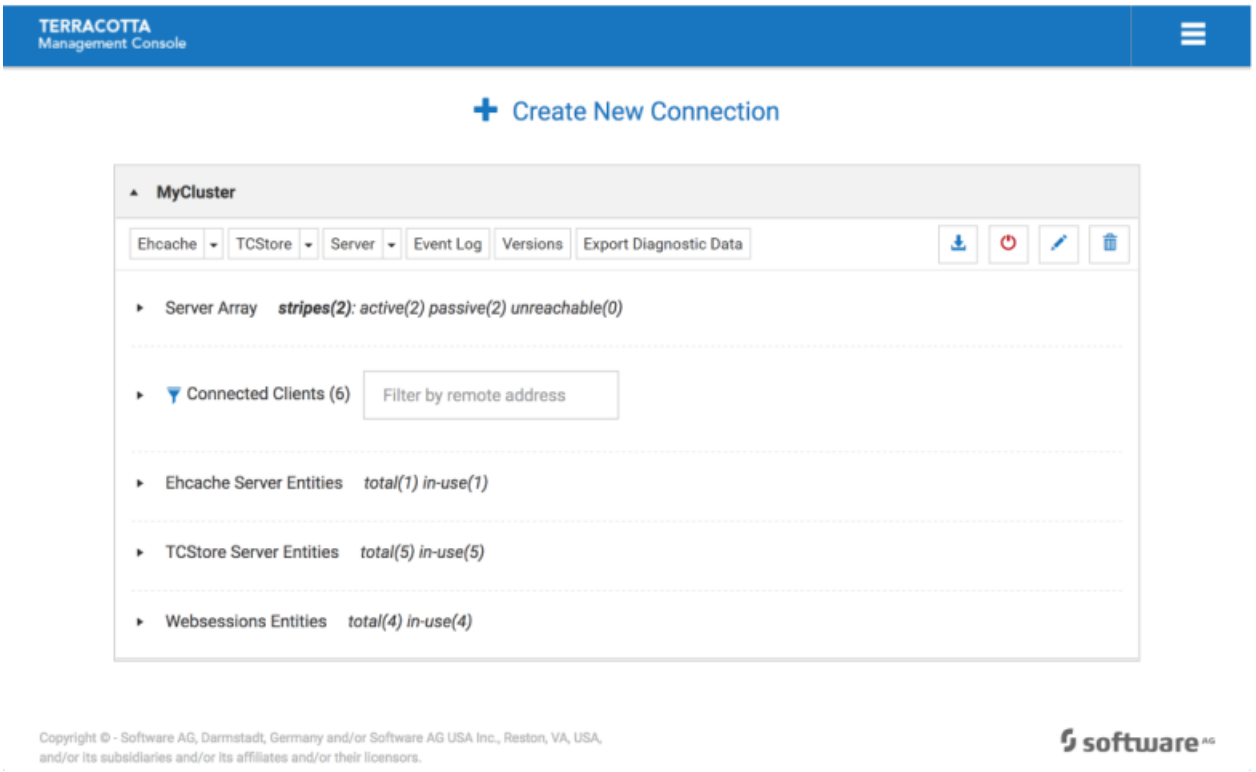
3. Enter the name you would like associated with this connection, for example “MyCluster”, then click **Next** to create the connection.

Screenshot: TMC Home Page, Connection Name

This connection name will be used to disambiguate multiple connections in the UI and when communicating with the TMS REST interface.

4. Upon completion of creating the new connection, a new *connection region* is shown on the Home Page, displaying the current state of your cluster.

Screenshot: TMC Home Page, State of the TSA after initial entries.



The connection to `MyCluster` in the example above shows a TSA comprised of 2 stripes, each containing a pair of Active-Passive servers, 4 currently connected clients, a single Ehcache server entity and a single TCStore server entity. Further detail can be exposed by drilling down into the display hierarchy.

Note:
For Terracotta Ehcache product users, the **TCStore** dropdown is deactivated.

Modifying a Connection's Properties

If you wish to modify your new connection's properties, use the *Edit this cluster* ( ) icon on the connection's header area.

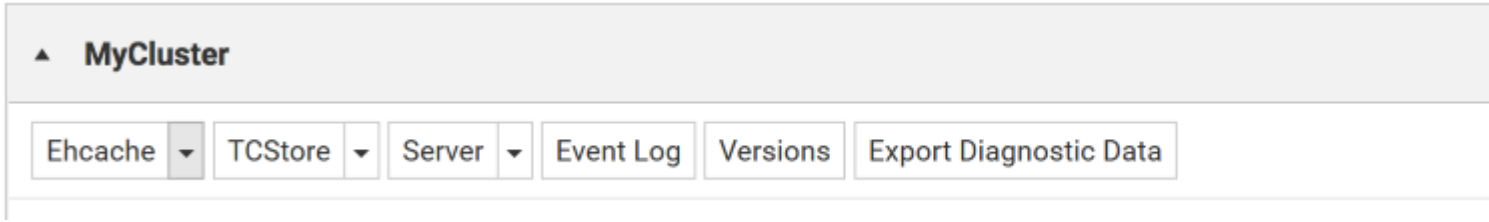
Screenshot: TMC Cluster Properties Editor (Overlay)

Connection Timeout	Timeout value, in seconds, when connecting to a cluster the first time or when reconnecting. Default is 10 seconds.
Read Timeout	Timeout value, in seconds, for all cluster communications. Default is 10 seconds.
Collector Interval	Collector interval for statistics. Default is 30 seconds. Minimum is 10 seconds and maximum is 60 seconds.

Deleting a Connection

To delete an existing connection use the `Delete this cluster` ( ) icon on the connection header area.

Screenshot: TMC Connection Area Header



Using the Configured Connections

The TMC home page shows each of your configured connections in its own collapsible region of the display that presents the totality of the cluster, allowing you to drill-down to different levels of hierarchy then jump to various detail views, such as for statistics or an entity's server-side resource usage.

Each configured connection region is comprised of several high-level facets which will now be described.

Buttons in the connection region header

The header of the connection region for a selected cluster contains the following selectable buttons:

- **Ehcache**
- **TCStore**
- **Server**
- **Events**
- **Export Diagnostic Data**

If you select the **Ehcache**, **TCStore**, **Server** or **Events** button, the **Detail** page will be displayed, with the appropriate tab (**Ehcache**, **TCStore**, **Server** or **Events**) already selected.

Each of these three buttons contains a dropdown menu. If you select an item from any of these dropdown menus, the **Detail** page will be displayed and the details for the selected dropdown entry will be in focus.

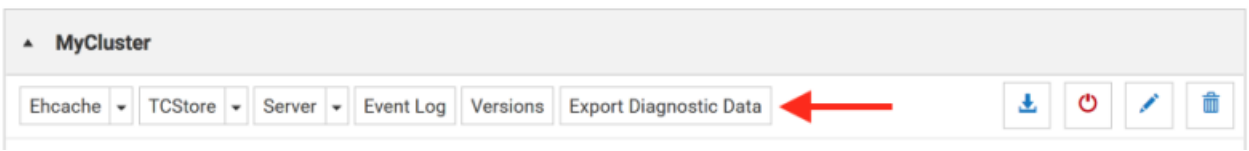
For details of the **Ehcache**, **TCStore**, **Server** and **Events** tabs of the **Detail** page, see the sections "Using the Ehcache Tab" on page 25, "Using the TCStore Tab" on page 35, "Using the Server Tab" on page 51 and "Using the Events Tab" on page 55.

For information about the button **Export Diagnostic Data**, see the following topic.

Export Diagnostic Data (Cluster level)

The **Export Diagnostic Data** button at the cluster level provides diagnostic information for every server in the cluster.

Screenshot: TMC Connection Area Header, Highlighting button for Export Diagnostic Data



The cluster level **Export Diagnostic Data** button exports the following diagnostic data as a zip file.

- Event Log - a csv file named `EventLog.csv`, containing cluster wide events. This can also be exported from the **Events** panel.

- Versions - a text file named `versions.txt`, containing build version information for all nodes in the cluster and their connected clients.
- For each server it also downloads seven diagnostic data files, which are explained in the next section, *Export Diagnostic Data (Server level)*.

The zip file is named using the format: `diagnostic-connectionName-dateTime.zip`

- e.g. `diagnostic-myClusterConnection-20170705221353.zip`

and the layout of the zip file is:

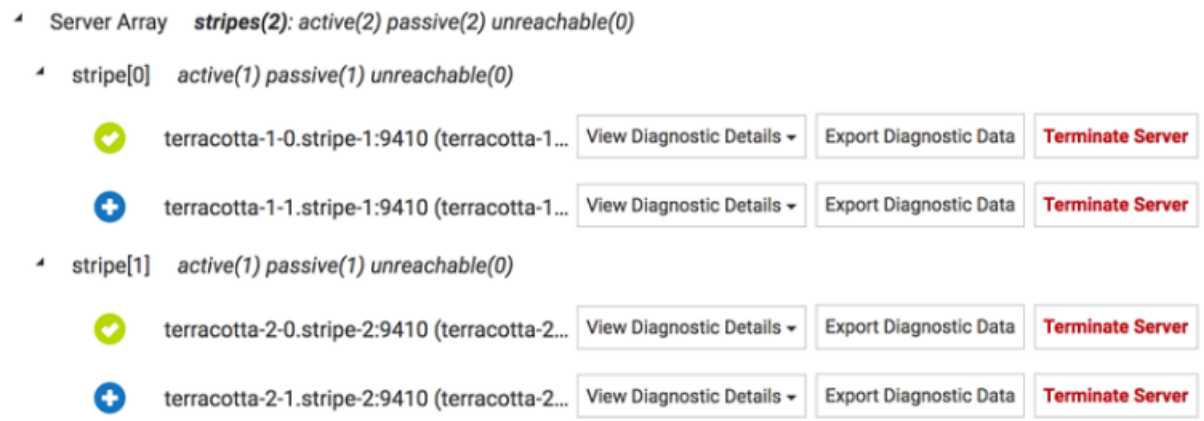
- one folder for each stripe, containing one folder for each server within the stripe
- each server folder contains the seven files referenced in the next section, *Export Diagnostic Data (Server level)*
- in the root of the zip file there is the cluster wide event log, `EventLog.csv`, and a `versions.txt` file which lists the versions of all nodes in the cluster and their connected clients.

Server Array

The *Terracotta Server Array* (TSA) is a collection of groups of servers, known as *stripes*. Servers within a stripe work together to provide *High-Availability*. If the *Active* server should fail, any one of the remaining *Passive* servers takes over as active. The *Active* server serves to (1) handle requests from clients to entities it contains and (2) to relay those client requests to each of the *Passive* servers. Servers in different stripes do not interact.

The TMC presents the current state of the server array, indicating the roles of each server. The following shows a server array consisting of two stripes, each containing two members, one *active* and one *passive*.

Screenshot: Example of a TMC Terracotta Server Array, two stripes, each having two members, one active, one passive.



The possible server states are:

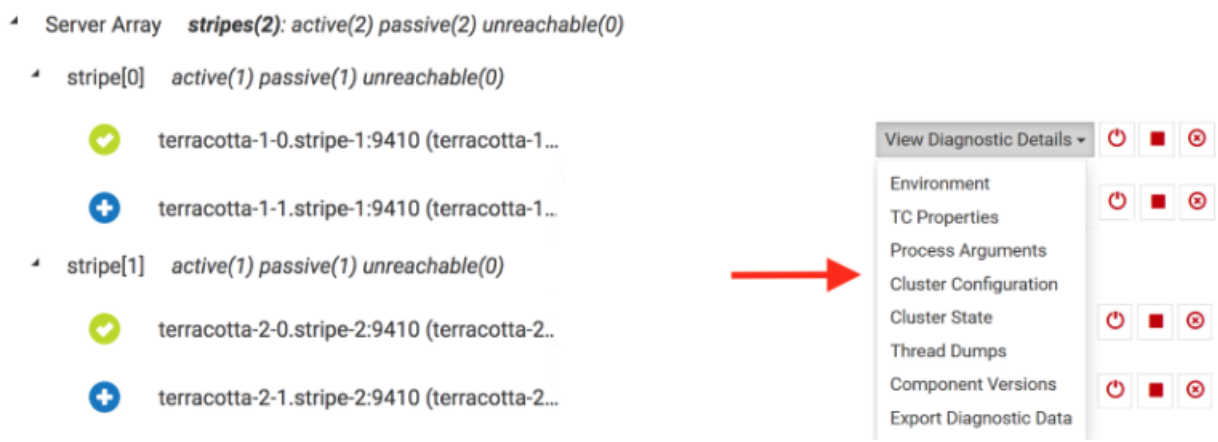
Icon	Server state	Description
	STARTING	server is starting
	UNINITIALIZED	server has started and is ready for election
	SYNCHRONIZING	server is synchronizing its data with the current active server
	PASSIVE	server is passive and ready for replicatio
	ACTIVE	server is active and ready to accept client
	ACTIVE_RECONNECTING	server is active but waits for previously known clients to rejoin before accepting new clients
	ACTIVE_SUSPENDED	server is active but blocked in the election process (consistency mode)
	PASSIVE_SUSPENDED	server is passive but blocked in the election process (consistency mode)
	START_SUSPENDED	server is starting but blocked in the election process (consistency mode)
	UNREACHABLE	server is unreachable from TM
	UNKNOWN	default server state shown in TMC before TMC is able to access the server stat

Diagnostic information for the members of your TSA can be downloaded as an archive file or viewed directly.

View Diagnostic Details (Server level)

The **View Diagnostic Details** dropdown at the server level lets you view directly the diagnostic information for that particular server. There is one diagnostic data dropdown per server and it is always positioned to the immediate right of the server name.

Screenshot: Example of a TMC Terracotta Server Array, dropdown highlighted in action.



Note:
Please don't confuse the stripe count with the stripe name. In this example there are two stripes, indicated by **stripes(2)**, and they are named `stripe[0]` and `stripe[1]`.

Export Diagnostic Data (Server level)

The **Export Diagnostic Data** button at the server level lets you download diagnostic information for that particular server in an archive file, similar in format to that provided at the cluster level.

The dropdown provides access to the seven diagnostic data files below. The file contents can be viewed/exported individually by selecting that option from the dropdown or by selecting the top level **Export Diagnostic Data** option which exports a zip file containing all of them.

1. Environment - shows a list of all the environment variables.
2. TC Properties - provides a list of all the TC config properties.
3. Process Arguments - displays all the command line arguments submitted for the process.
4. Cluster Configuration - shows the Terracotta configuration file.
5. Cluster State - displays information on the current cluster state.
6. Thread Dumps - exports the thread dump as a txt file.
7. Component Versions - displays a listing of build version information for all servers and their connected clients.

Tip:
What is a thread dump? A Java thread dump is a way of finding out what every thread in the JVM is doing at a particular point in time. This is especially useful if your Java application sometimes seems to hang when running under load, as an analysis of the dump will show where the threads are stuck.

The zip file is named using the format:
`diagnostic-connectionName-stripeName-serverName-dateTime.zip`

- e.g. `diagnostic-myClusterConnection-stripe[0]-testServer0-20170705221356.zip`

and the layout of the zip file is one folder, named after the server name, containing the 6 data files above.

Connected Clients

In the Terracotta Platform, a *client* is an application end-point. In your application, the clustered `CacheManager` that is configured and initialized **is** a Terracotta client. Each client maintains a connection to the active server in each stripe. In general, anything that connects to a server is considered a client.

For more information, see section *Terracotta Server Administration Guide* > *Clients in a Cluster*.

Each client has a *Client Identifer* that serves to uniquely identify that client. The form of the identifier is:

```
<pid>@<ip-addr>:<client-type>:<server-entity-name>:<uuid>
```

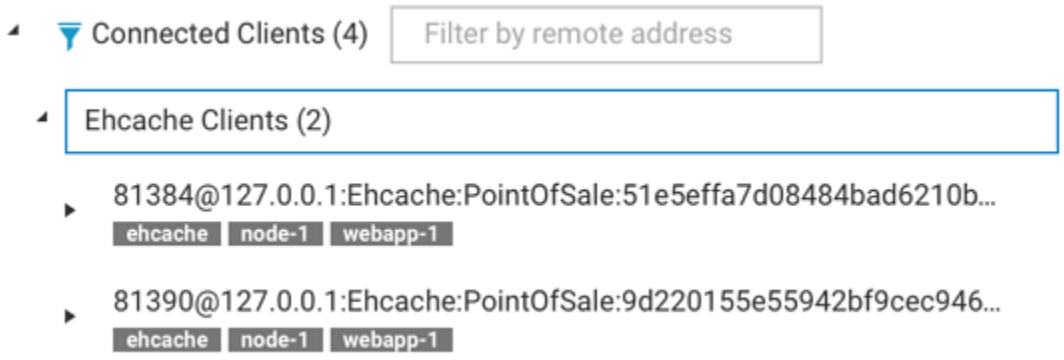
where:

<pid>	the *process identifier* of the Java Virtual Machine hosting this client
<ip-addr>	the *IP address* of the machine hosting this JVM
<client-type>	[Ehcache | Store | Unknown]
<server-entity-name>	the name of the server-side entity the client is connected to
<uuid>	a unique identifier that serves to disambiguate clients in the same JVM, accessing the same server entity

- **Ehcache Clients**

  The following shows two caching clients.

  Screenshot: Example Connected Clients: Two caching clients.

  The input field located next to *Connected Clients* serves to show only those clients whose identifier contains the entered value. In the example above, entering 81390 (the *process identifier* of the 2nd client) would filter out the first client.

  Filtering Rules

  - accepts a space-separated list of terms to match (prefix with ! to negate)
  - terms can apply to any components of the *Client Identifier* or any supplied tags

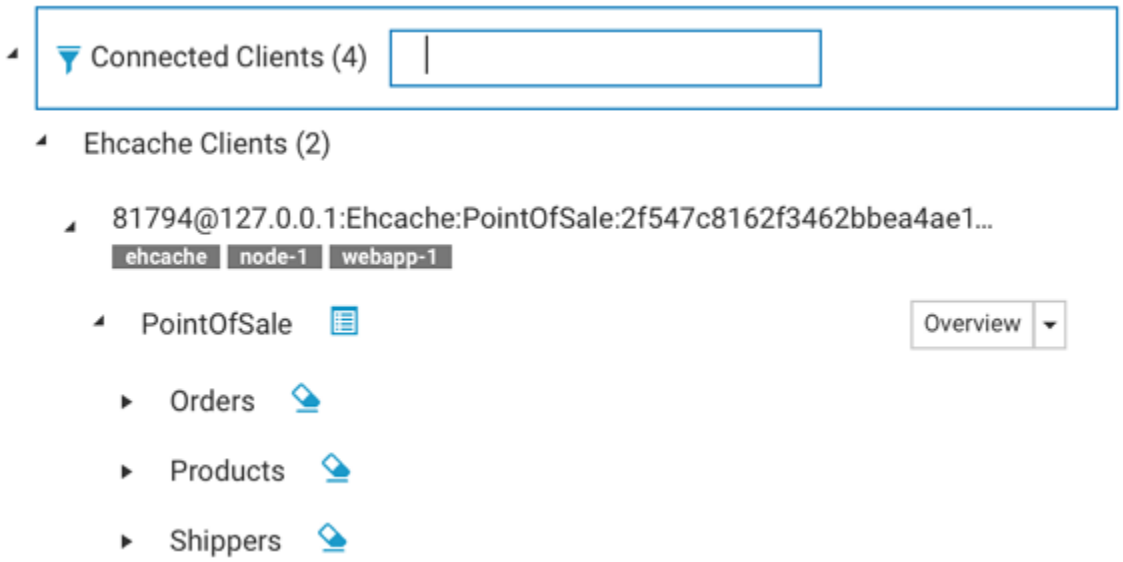
- negated terms must not match any of the above

- clear the input field content or click the toggle button ( ) to show all clients

Under each identifier is an optional set of user-defined tags, specified in the CacheManager's configuration.

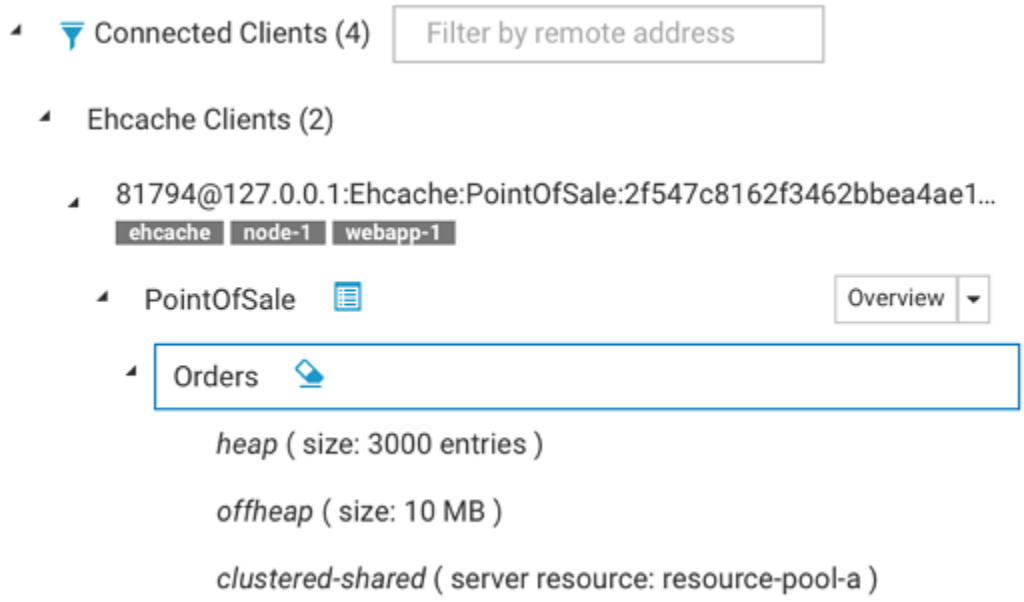
Expanding a *caching client* exposes the CacheManager's alias, which is defined in the configuration. The dropdown lets you jump to various detail views, pre-selecting this client and CacheManager in those views. View the CacheManager's configuration by selecting the **configuration** ( ) icon. The cache entries can be cleared by selecting the **clear cache** ( ) icon.

Screenshot: TMC Cache Manager's Configuration, elements collapsed



Expanding the cache shows important configuration elements related to that cache.

Screenshot: TMC Caching ClientExpanded



- **TCStore Clients**

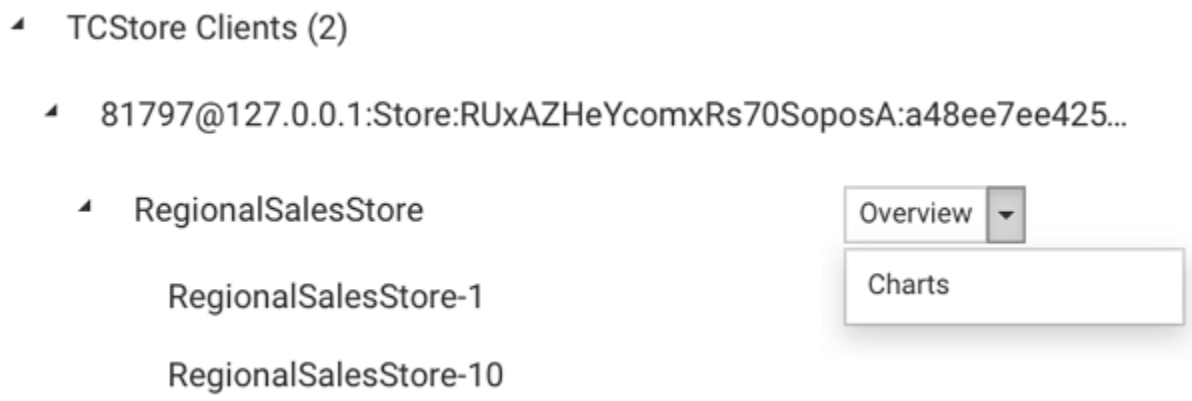
  Within the connected clients section there is also a subsection for all the dataset clients.

  Expanding the **TCStore Clients** section provides a list of all the connected clients, ordered by client identifiers.

  Further expanding a client identifier shows all datasets associated with that particular client and for each dataset there is a dropdown to navigate quickly to its overview or chart statistics.

  Lastly when expanding each dataset there is also a list of all its dataset instances

  Screenshot: Expanding a Client Identifier



Ehcache Entities

The Terracotta Platform consists of client-side programmatic (API) artifacts and server-side entities that work together to provide highly-available, performant, distributed data access. In

your application you configure a CacheManager, both on the client- and server-side. The client-side configuration relates to such things as the maximum OnHeap size-in-bytes for a particular cache. The server-side configuration relates to the remote storage tier that is used to store your cache entries.

Each of your *Ehcache Clients* communicates with its remote storage tier when executing normal cache operations, such as putting an entry into the cache.

Ehcache clients configured to use the same remote storage tier are effectively sharing access to the same cache data.

TCStore Entities

A TCStore client is essentially a connection to a clustered or embedded Dataset.

A single client instance can *fetch* an arbitrary number of *handles* to the underlying Dataset, referred to as a Dataset instance. Operations statistics are maintained on a per-instance basis.

3 Using the Ehcache Tab

The **Ehcache** tab of the **Detail** page contains detailed presentations relating to your caching clients. The **Detail** page is selectable via buttons in the home page.

You can view:

- Overview statistics concerning this moment in time, in grid form. Select this view by clicking the icon .
- Historical statistics, in chart form. Select this view by clicking the icon .
- Cache size information. Select this view by clicking the icon .

Your caching client automatically gathers and periodically sends to the TMS low-level statistics concerning counters and sizing. From those low-level statistics the TMS can synthesize a variety of derived statistics, such as rates (HitRate) and ratios (HitRatio).

The following categories of statistics are available:

Category	Description
Cache	encompasses all the following caching tiers
OnHeap	the JVM heap tier
OffHeap	the JVM OffHeap (direct memory) tier
Disk	the disk tier
Clustered	the cluster tier

The following raw counter values, as well as associated rate of change, are available for each category listed above:

Counter	Description
<Category>:PutCount	number of times your application has put a new entry into the cache
<Category>:UpdateCount	number of times your application has put a new entry into the cache that replaced an existing entry (same key)

Counter	Description
<Category>:RemovalCount	number of times your application explicitly removed an entry from the cache
<Category>:ExpirationCount	number of entries removed from the cache due to an expiration policy (time-to-live, etc.)
<Category>:EvictionCount	number of entries removed from the cache due to space constraints
<Category>:HitCount	number of gets that returned an existing entry
<Category>:HitRatio	ratio of hits to gets
<Category>:MissCount	number of gets that did not return an entry
<Category>:MissRatio	ratio of misses to gets

The following category-specific raw counters are also available:

Counter	Description
Disk:MappingCount	number of entries stored on disk
Disk:AllocatedByteSize	number of bytes allocated for storage. AvailableByteSize = AllocatedByteSize - OccupiedByteSize
Disk:OccupiedByteSize	total size of all entries stored on disk
OffHeap:MappingCount	number of entries stored in direct memory
OffHeap:AllocatedByteSize	number of bytes allocated off heap in memory. AvailableByteSize = AllocatedByteSize - OccupiedByteSize
OffHeap:OccupiedByteSize	total size of all entries stored in direct memory
OnHeap:MappingCount	number of entries that store in regular heap memory
OnHeap:OccupiedByteSize	total size of all entries store in regular heap memory

The following latencies are also available:

Counter	Description
Cache:GetHitLatency	latencies of get operations leading to a cache hit. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Cache:GetHitLatency#50 ■ Cache:GetHitLatency#95

Counter	Description
	■ Cache:GetHitLatency#99 ■ Cache:GetHitLatency#100
Cache:GetMissLatency	latencies of get operations leading to a cache miss. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Cache:GetMissLatency#50 ■ Cache:GetMissLatency#95 ■ Cache:GetMissLatency#99 ■ Cache:GetMissLatency#100
Cache:PutLatency	latencies of successful put operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Cache:PutLatency#50 ■ Cache:PutLatency#95 ■ Cache:PutLatency#99 ■ Cache:PutLatency#100
Cache:GetRemoveLatency	latencies of successful remove operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Cache:RemoveLatency#50 ■ Cache:RemoveLatency#95 ■ Cache:RemoveLatency#99 ■ Cache:RemoveLatency#100

Note:
Ratios and Rates

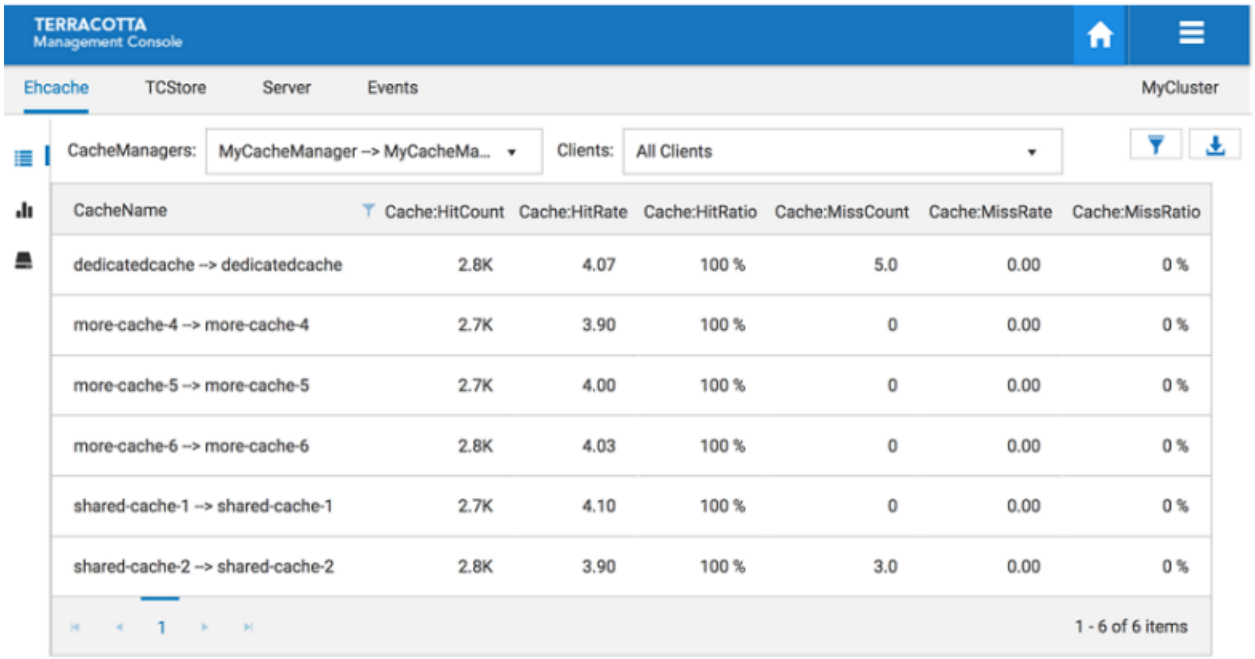
The TMS gathers raw counters from your application and the servers. Rates of change (rates) and ratios are synthesized based on windowed sample aggregation. The aggregation method depends on the statistic type. For counters and latencies, the maximum value observed during an interval of time is displayed. The TMC requests a different number of samples depending on the chart's time-frame, which will determine the length of the window to use for rates and ratio calculation. The units for rates are always operations per second. Ratios are unitless.

Overview Panel

The Overview Panel lets you view statistics concerning what is happening right now (or just a while ago) with your caching client, broken out by cache. By default the statistics are aggregated

across all clients using the same CacheManager. Any particular caching client can also be selected for viewing.

Screenshot: TMC Caching Overview, listing caches and statistics.



Use the **CacheManagers** dropdown to select the aliased CacheManager for which to show statistics. The entries in this dropdown are of the form *CacheManager Alias* > *Storage Handle* due to the fact that different client-side CacheManagers can be configured to use the same storage on the TSA. Furthermore, CacheManagers that are otherwise identical can be configured with different aliases. We show the mapping here so you have a chance to disambiguate CacheManagers should you choose to configure them in this way.

Use the **Clients** dropdown to select a particular caching client for viewing. By default, statistics are aggregated across all clients of the selected CacheManager.

Additional Grid Features

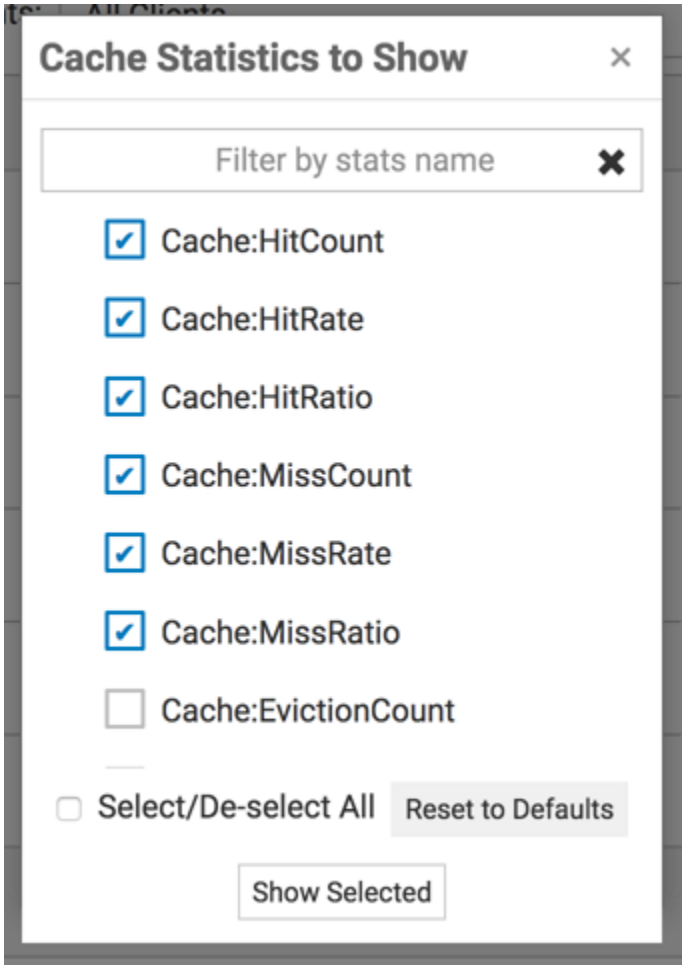
- Temporarily re-arrange grid columns via drag-and-drop
- Sort on columns or filter on CacheName
- Scroll horizontally if the grid columns overflow the available space

Use the **Export** ( ) icon to download a spreadsheet of the currently displayed values.

Use the **Filter cache statistics** ( ) icon to select which statistics to display. The set of statistics to view defaults to **Cache**-level statistics that are applicable to all use-cases.

You can also filter statistics by name, using space-separated terms.

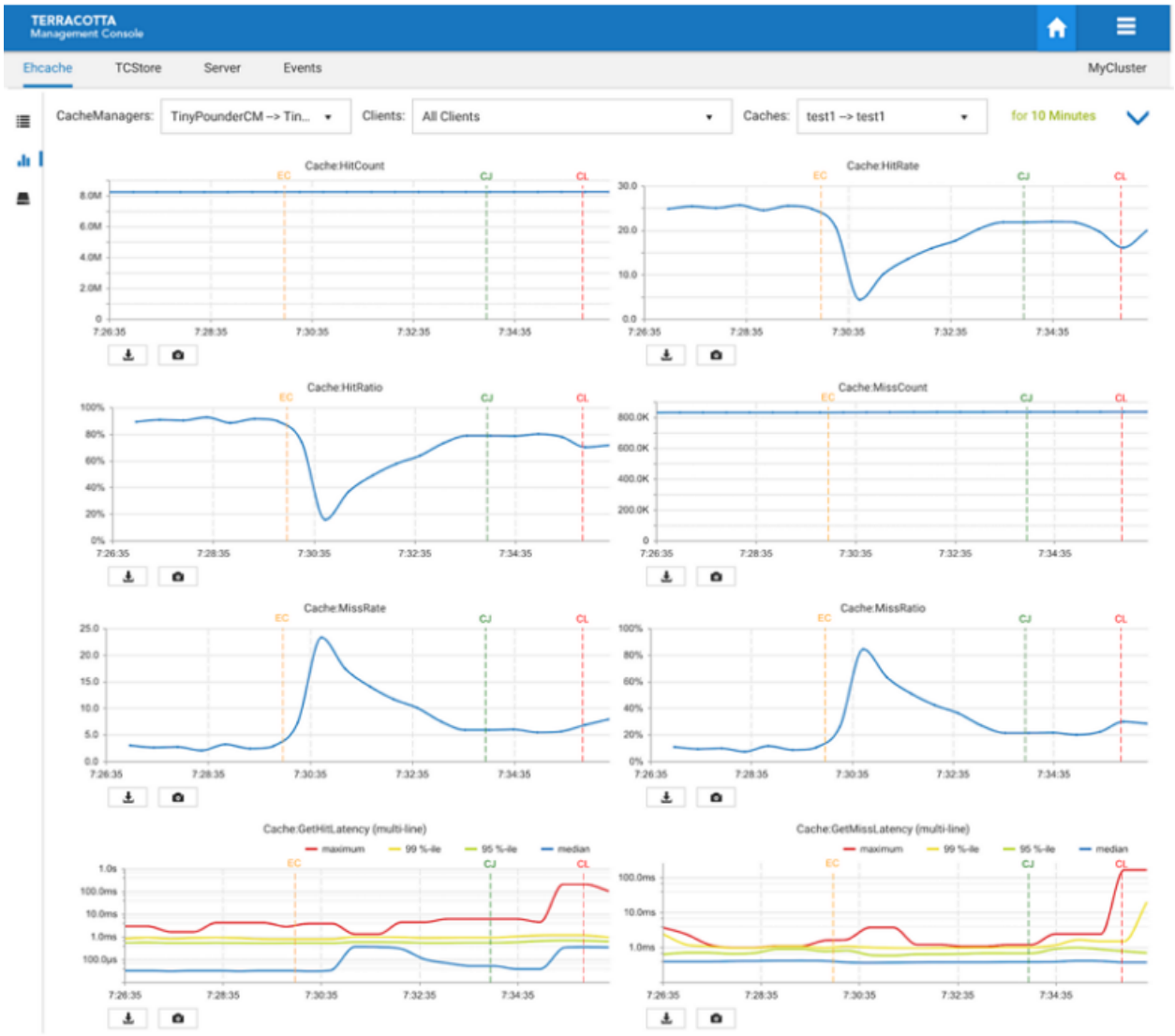
Screenshot: Overlay, choosing statistics



Charts Panel

The **Charts** Panel lets you view statistics over the past 5 minutes. The TMC requests 30 samples be returned. Unlike the **Overview** Panel, which shows all the caches contained by the selected CacheManager, the **Charts** Panel shows statistics for particular caches.

Screenshot: Charts Panel



Use the **CacheManagers** dropdown to select the aliased CacheManager for which to show statistics. The entries in this dropdown are of the form **CacheManager Alias > Storage Handle** due to the fact that different client-side CacheManagers can be configured to use the same storage on the TSA. Furthermore, CacheManagers that are otherwise identical can be configured with different aliases. We show the mapping here so you have a chance to disambiguate CacheManagers should you choose to configure them in this way.

Use the **Clients** dropdown to select a particular caching client for viewing. By default, statistics are aggregated across all clients of the selected CacheManager.

Use the **Caches** dropdown to select a particular cache for viewing. By default, the first cache listed is selected.

Use the slider (

) to set how many columns of charts you would like displayed.

Use the events checkbox (Events) to choose whether or not you want to display clients and cache cleared events on the charts. This causes additional dotted lines to appear on the charts, showing the time when the following events occurred: "EC" = Ehcache Cleared, "CJ" = Client Joined, "CL" = Client Left.

Use the **Filter cache statistics** ( ) icon to select which statistics to display. The set of statistics to view defaults to **Cache**-level statistics that are applicable to all use-cases.

Use the **Take a snapshot of all charts** ( ) icon to download a single PNG file containing the current values of all displayed charts.

Directly under each individual chart, use the **Export to PDF** ( ) icon or the **Export to PNG image** ( ) icon to download the chart in the selected format.

Additional Chart Features

- Rearrange charts via drag-and-drop; the order is preserved until the browser is closed or its cache is manually cleared. If new statistics are selected, they will appear after the charts, and then you can rearrange them.
- Double-click a chart to get an enlarged snapshot
- Check the box named "Events" to display "Client Joined" , "Client Left" and "Ehcache Cleared" events on your charts

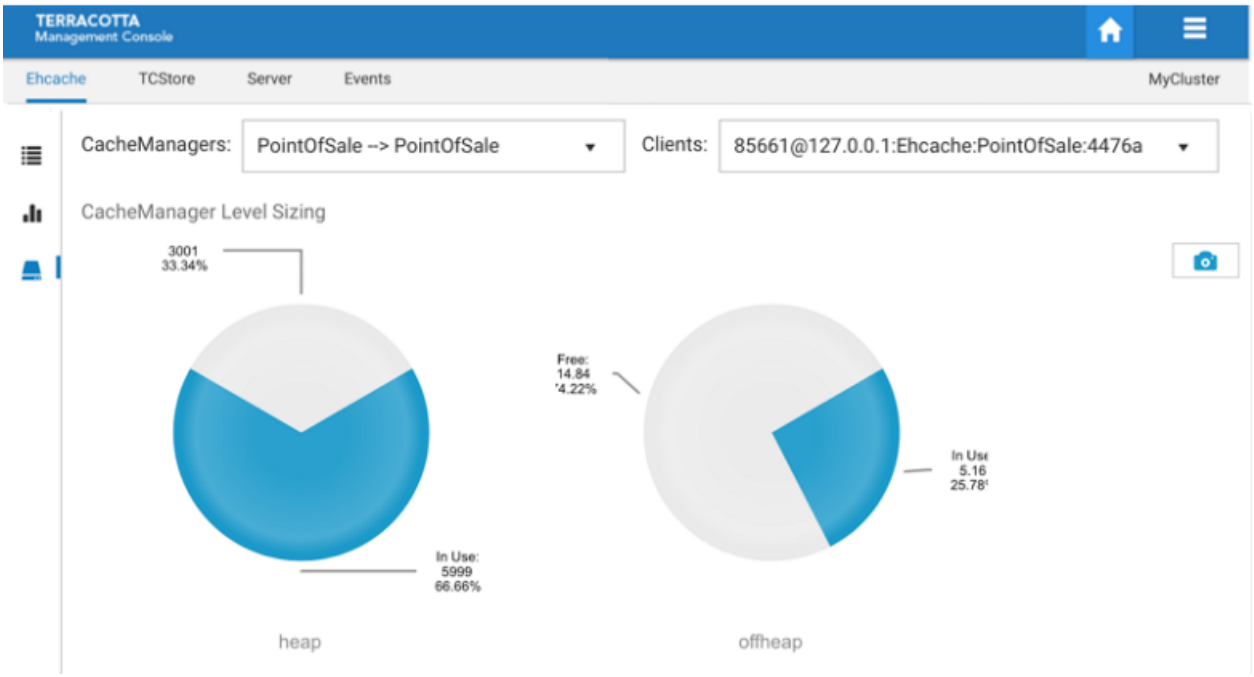
Sizing Panel

The Sizing Panel shows how much space is being used by your CacheManagers and caches across the different local storage tiers you've configured. Sizing information related to the *Clustered Storage Tier* can be accessed via the **Remote** buttons.

The **Sizing** Panel is composed of two sections: *CacheManager Level Sizing* and *Cache Level Sizing*.

- **CacheManager Level Sizing**

  Screenshot: Sizing Panel



Shown is a pie chart for each local storage tier you've configured for use by your caches, displaying the amount of the available storage that is being used.

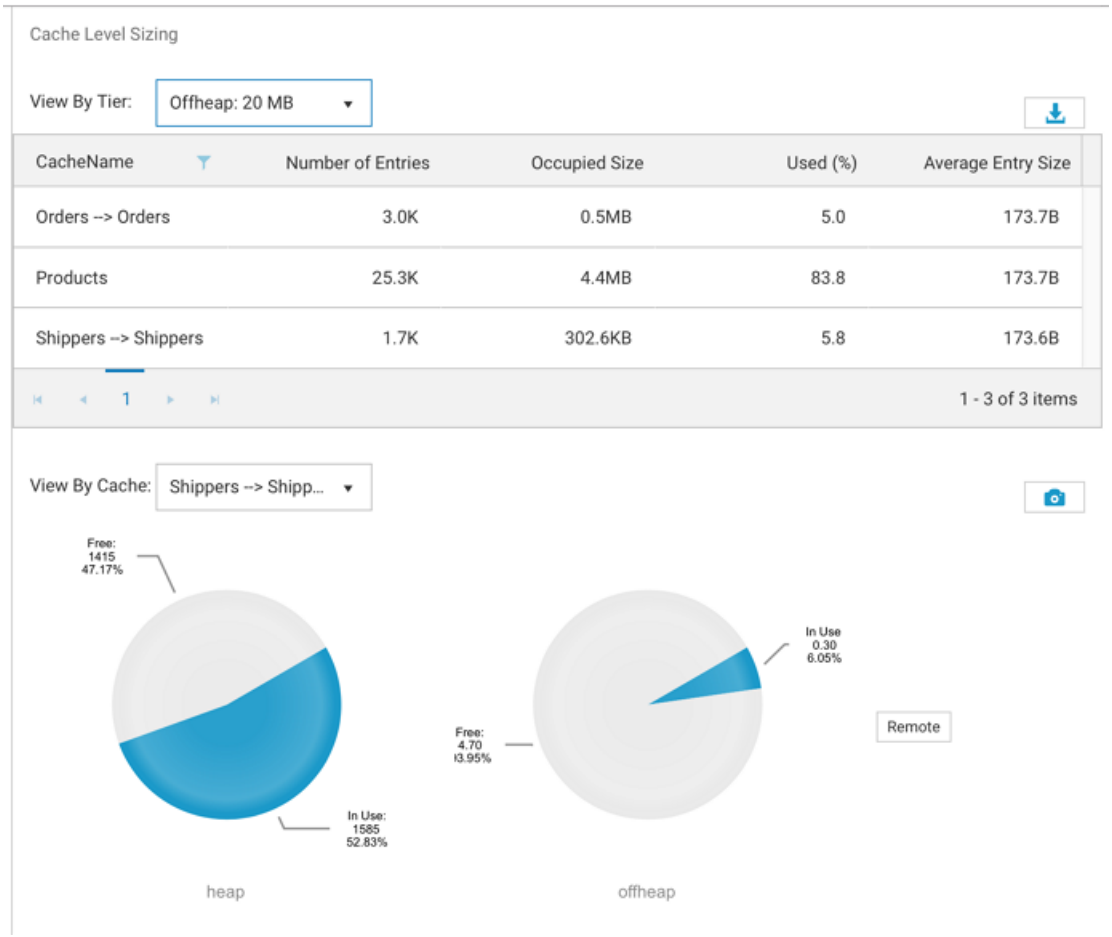
Use the **CacheManagers** dropdown to select the aliased CacheManager for which to display sizing information.

Use the **Clients** dropdown to select a particular caching client for which to display sizing information. By default the first client listed is selected.

Use the **Remote** button to navigate to the Resource Usage panel to view how the server-side caching entity is making use of the server's configured resources, such as OffHeap storage. See the section "Resource Usage Panel" on page 51 for related information.

- **Cache Level Sizing**

  Screenshot: Cache Level Sizing



Use the **View By Tier** dropdown to view tier-specific sizing information for all caches contained by the selected CacheManager in grid form.

Note:

Byte-sizing Limitations

Cache tiers that are sized by *entries* cannot provide *Occupied Size* or *Average Entry Size* and are displayed as `N/A` for *Not Applicable*.

Use the **View By Cache** dropdown to view the tier usage breakdown for a particular cache, in pie chart form.

4 Using the TCStore Tab

The **TCStore** tab of the **Detail** page opens a view containing detailed statistics for TCStore dataset operations. The **Detail** page is selectable via buttons in the home page.

The following raw counter values, as well as associated rate of change, are available:

Counter	Description
Add:AlreadyExists	count of adds that failed due to a record with the specified key already existing in the dataset
Add:Failure	count of unsuccessful additions to a dataset
Add:Success	count of successful additions to a dataset
Delete:Failure	count of unsuccessful deletions from a dataset
Delete:NotFound	count of deletes that failed due to a record with the specified key not existing in the dataset
Delete:Success	count of successful deletions from a dataset
Get:Failure	count of unsuccessful gets from a dataset
Get:NotFound	count of failed gets that were due to no record with the specified key existing in the dataset
Get:Success	count of successful gets from a dataset
Update:Failure	count of unsuccessful attempts to update a record in the dataset
Update:NotFound	count of update failures due to no record with the specified key existing in the dataset
Update:Success	count of successful record updates
Stream:Request	count of record streams obtained from the dataset
Stream:Failure	count of unsuccessful stream requests

Note:

Each of the statistics listed above is a simple counter but each has an associated rate, for instance, `Get:Success:Rate`, whose units are gets per second.

The following latencies are also available:

Dataset:GetLatency	latencies of get operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Dataset:GetLatency#50 ■ Dataset:GetLatency#95 ■ Dataset:GetLatency#99 ■ Dataset:GetLatency#100
Dataset:AddLatency	latencies of add operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Dataset:AddLatency#50 ■ Dataset:AddLatency#95 ■ Dataset:AddLatency#99 ■ Dataset:AddLatency#100
Dataset:UpdateLatency	latencies of update operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Dataset:UpdateLatency#50 ■ Dataset:UpdateLatency#95 ■ Dataset:UpdateLatency#99 ■ Dataset:UpdateLatency#100
Dataset:DeleteLatency	latencies of delete operations. 4 different percentiles are returned: median, 95th, 99th and maximum. ■ Dataset:DeleteLatency#50 ■ Dataset:DeleteLatency#95 ■ Dataset:DeleteLatency#99 ■ Dataset:DeleteLatency#100

These dataset operation statistics are sent to the TMS periodically by each dataset client and then are available to view in the TMC.

There are two TMC panels that show visualizations of the dataset operation statistics:

- Overview panel: displays the latest statistics for datasets and dataset instances in a tabular layout.
- Charts panel: presents a historical view of dataset statistics over a period of time via a graphical layout.

Note:

A dataset (e.g. `dataset1`) can have multiple instances and you can distinguish each dataset instance by its name, which will end in a dash '-' followed by a number. Thus an instance of `dataset1` could be named: `dataset1-1`.

The TCStore Overview Panel

The TCStore overview panel allows you to see the latest real time statistics for all of your datasets. In addition, the datasets can be filtered by:

- Dataset name
- Clients (Dataset instances in your application)

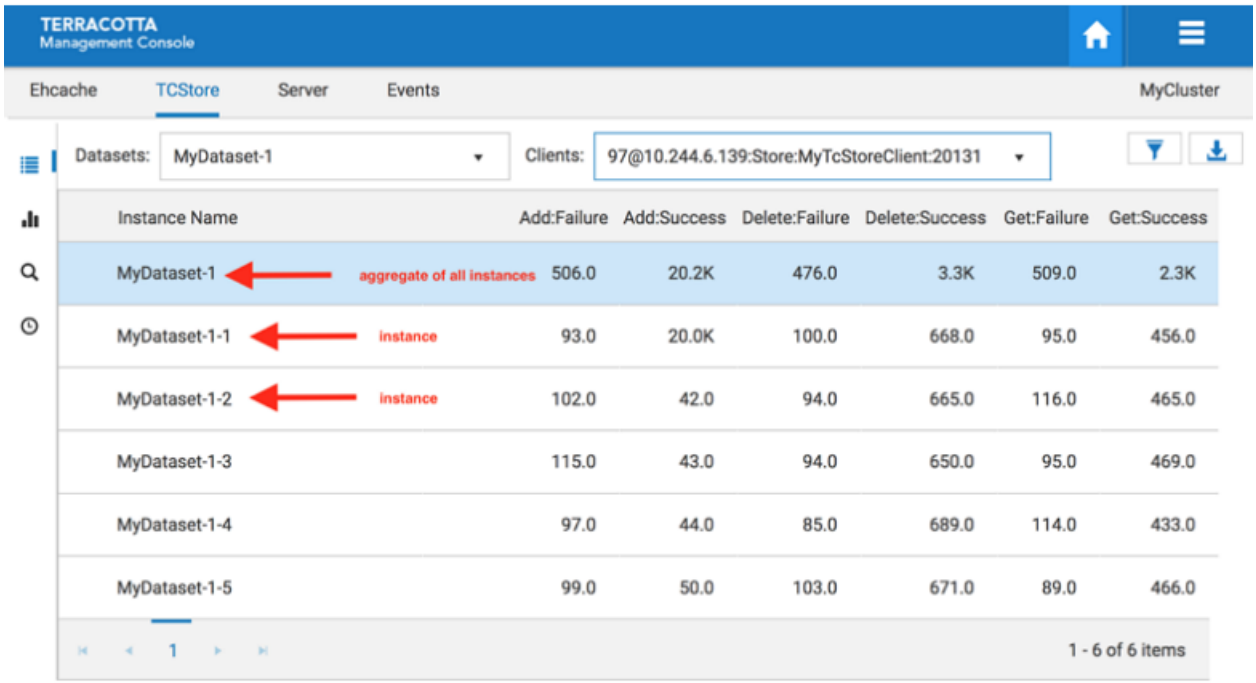
Note:

Note that the top highlighted dataset row is an aggregation of all the dataset instance statistics.

Filter by Dataset Name

The **Datasets** dropdown option allows you to view a selected dataset. The resulting view will show all dataset instances for the selected dataset.

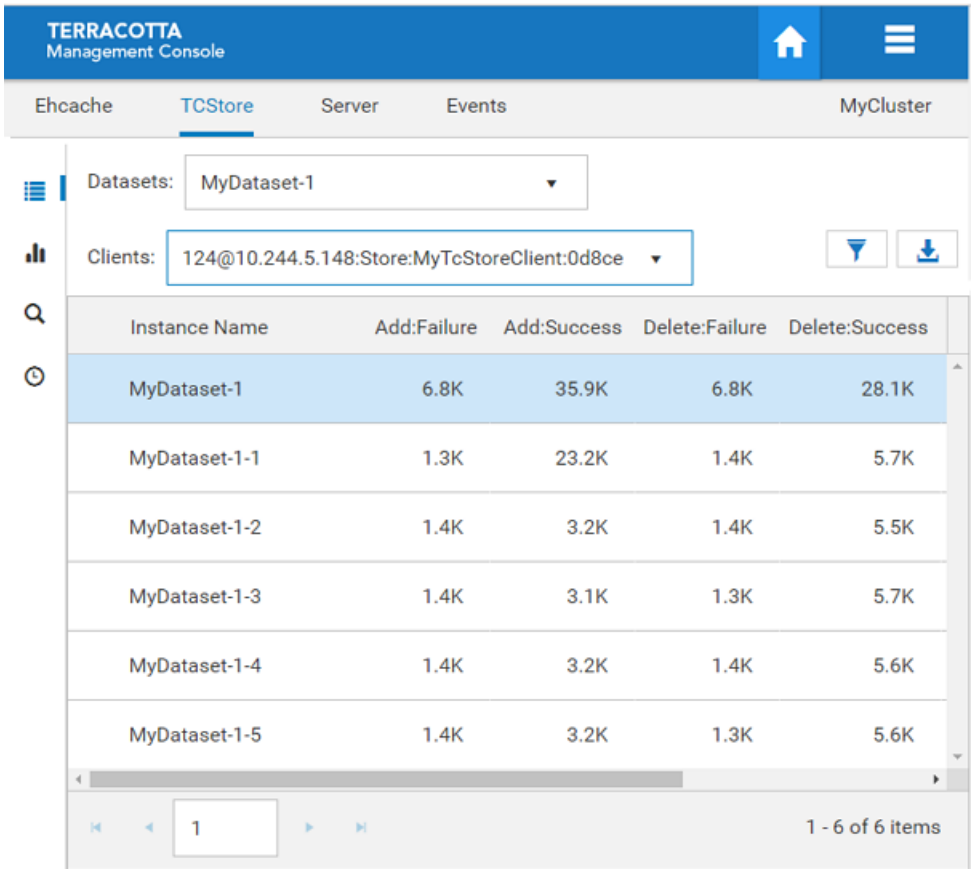
Screenshot: TCStore Overview Panel, Datasets dropdown.



Filter by a single client

The **Clients** dropdown option allows you to filter by a particular client or all clients. When selecting a single client the result will only include the dataset and dataset instances of the chosen client.

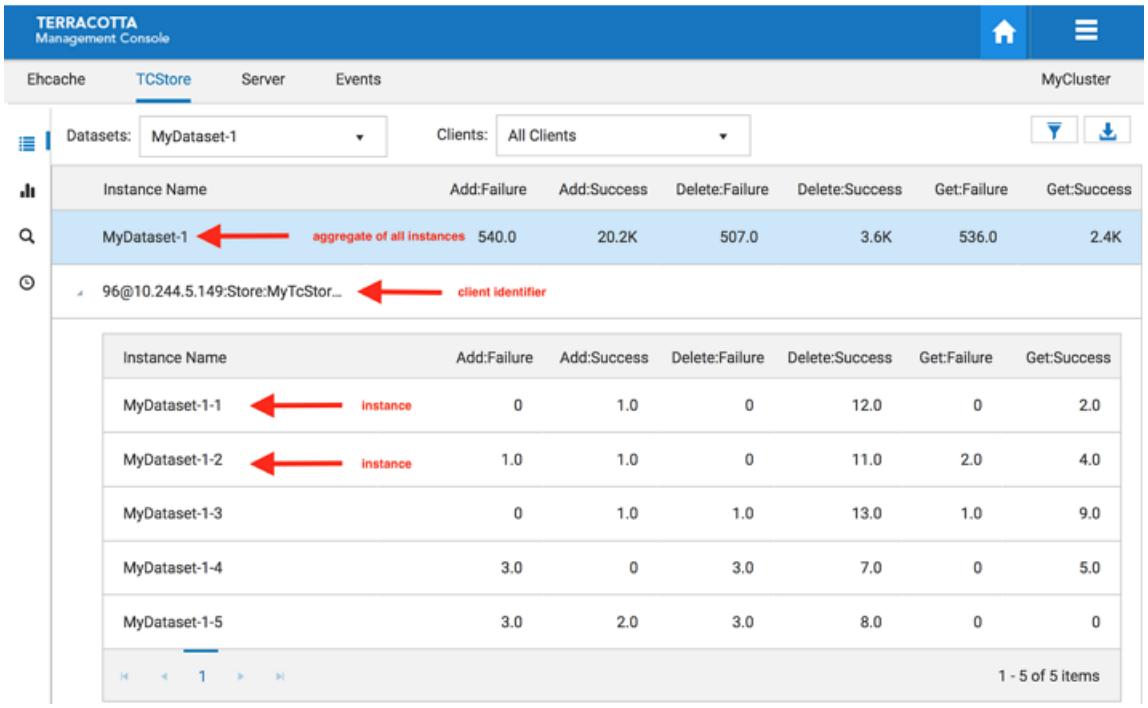
Screenshot: TCStore Overview Panel, Clients dropdown.



Filter by All Clients

When selecting all clients from the dropdown list the view will slightly change to show all clients for a particular dataset. In this layout the highlighted top row shows the aggregated statistics from all clients for the selected dataset instance. The row directly below the aggregated dataset statistics displays the client identifier, which can be expanded to show statistics for every dataset instance on that client.

Screenshot: TCStore Overview Panel, all clients of a particular dataset.



Note:

Additional Grid Features

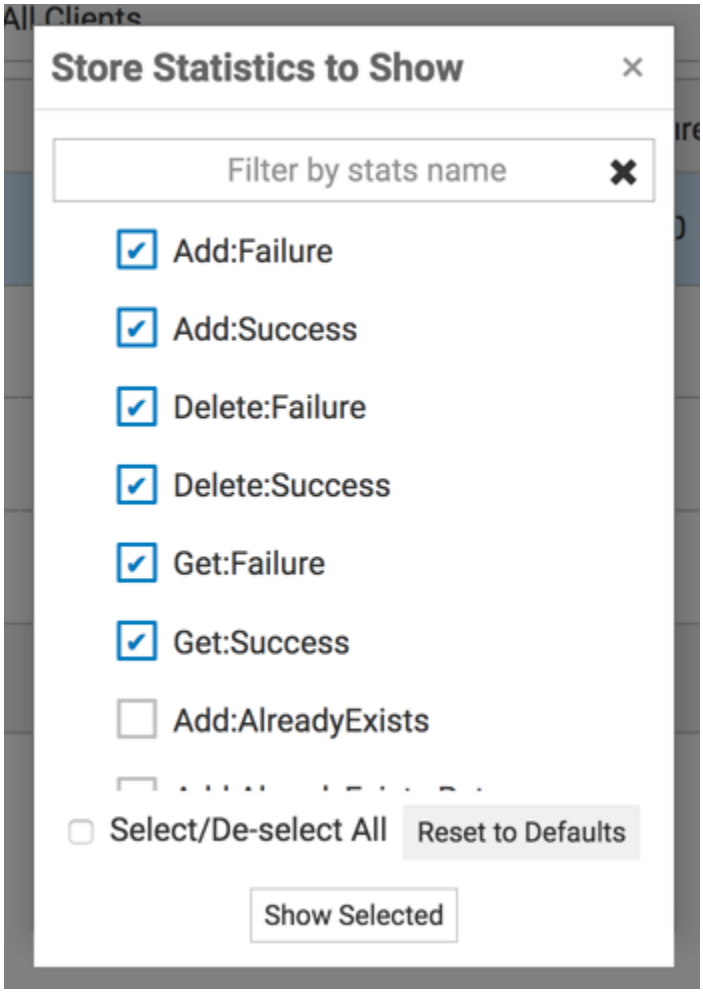
- Temporarily re-arrange grid columns using drag-and-drop
- Sort on columns
- Scroll horizontally if the grid columns overflow the available space

Use the Export to Excel ( ) icon to download a spreadsheet of the currently displayed values.

Use the Filter dataset statistics ( ) icon to select which statistics to display.

You can also filter stats by name, using space-separated terms.

Screenshot: Overlay, dataset statistics to show



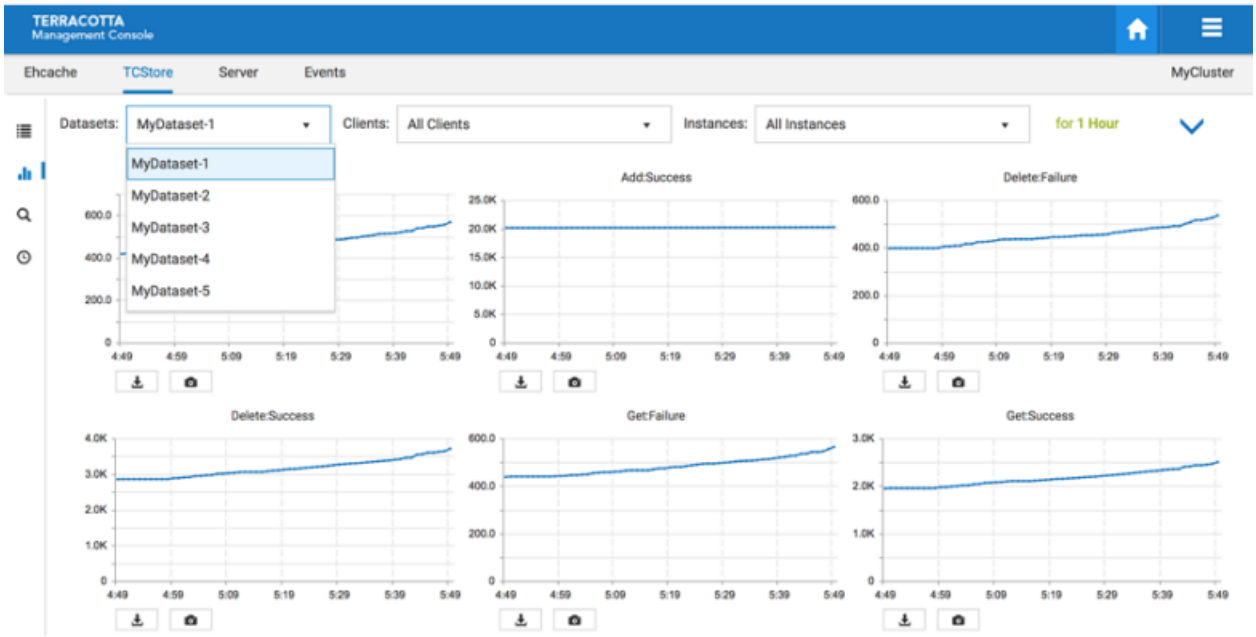
The TCStore Charts Panel

The charts panel allows you to view dataset statistics over a period of time. Each statistic is represented in its own chart which shows its values over the last 5 minutes. This 5 minute window also constantly updates to ensure always seeing the most recent historical statistics.

Filter by dataset

The historical view can be filtered by a particular dataset, which is useful because a dataset could exist on more than one client.

Screenshot: TCStore Overview Panel, Historical View filtered by a particular dataset.

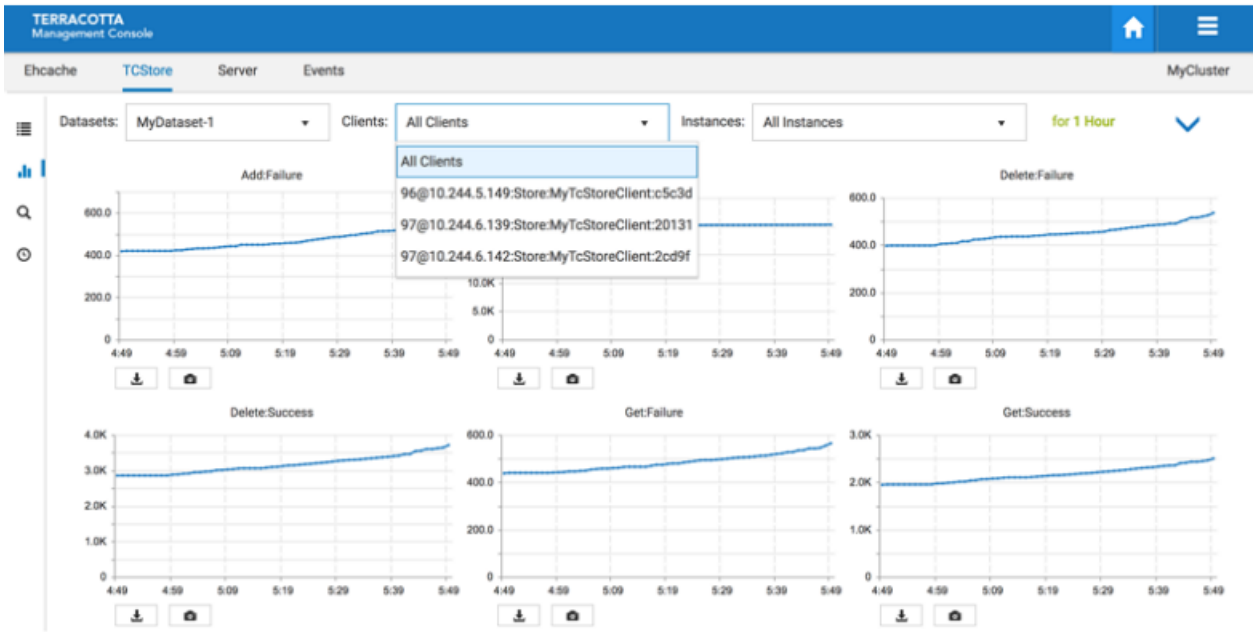


Filter by client

There is also an option to filter the history by a specific client or all clients.

- one client - you only see dataset statistics for the selected client, which is represented as a client identifier in the dropdown
- all clients - the dataset instance statistics are aggregated across all the clients

Screenshot: TCStore Overview Panel, Historical View filtered by clients.

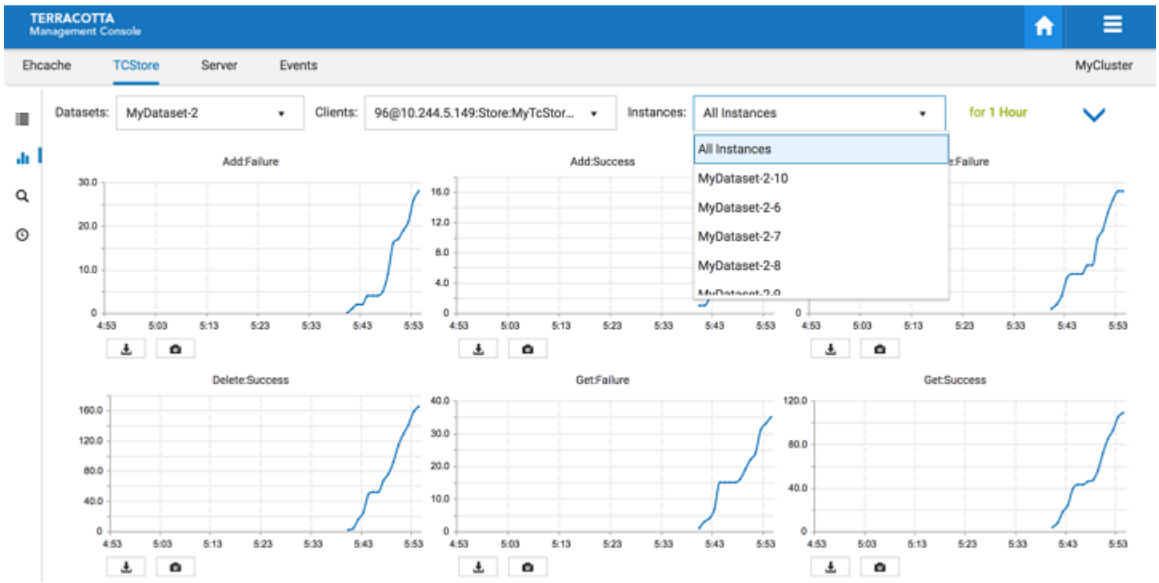


Filter by instances

You can filter by dataset instances, either by a specific instance or all instances.

- one instance - this only shows statistics for the selected dataset instance
- all instances - this selection aggregates all dataset instance values for the selected dataset

Screenshot: TCStore Overview Panel, Historical View filtered by clients.



Use the slider (2 3 4) to set how many columns of charts you would like displayed.

Use the **Filter dataset statistics** ( ) icon to select which statistics to display.

Use the **Take a snapshot of all charts** ( ) icon to download a single PNG file containing the current values of all displayed charts.

Directly under each individual chart, use the **Export to PDF** ( ) icon or the **Export to PNG image** ( ) icon to download the chart in the selected format.

The TCStore Explorer Panel (Ad-hoc Query)

This panel is designed to run ad-hoc queries against a selected dataset in the cluster. It is not meant to replace programming Stream queries or the SQL query API. The panel's typical use case is: When you need to verify the existence of a record in an ad-hoc manner, you can run simple queries in this panel and check the result.

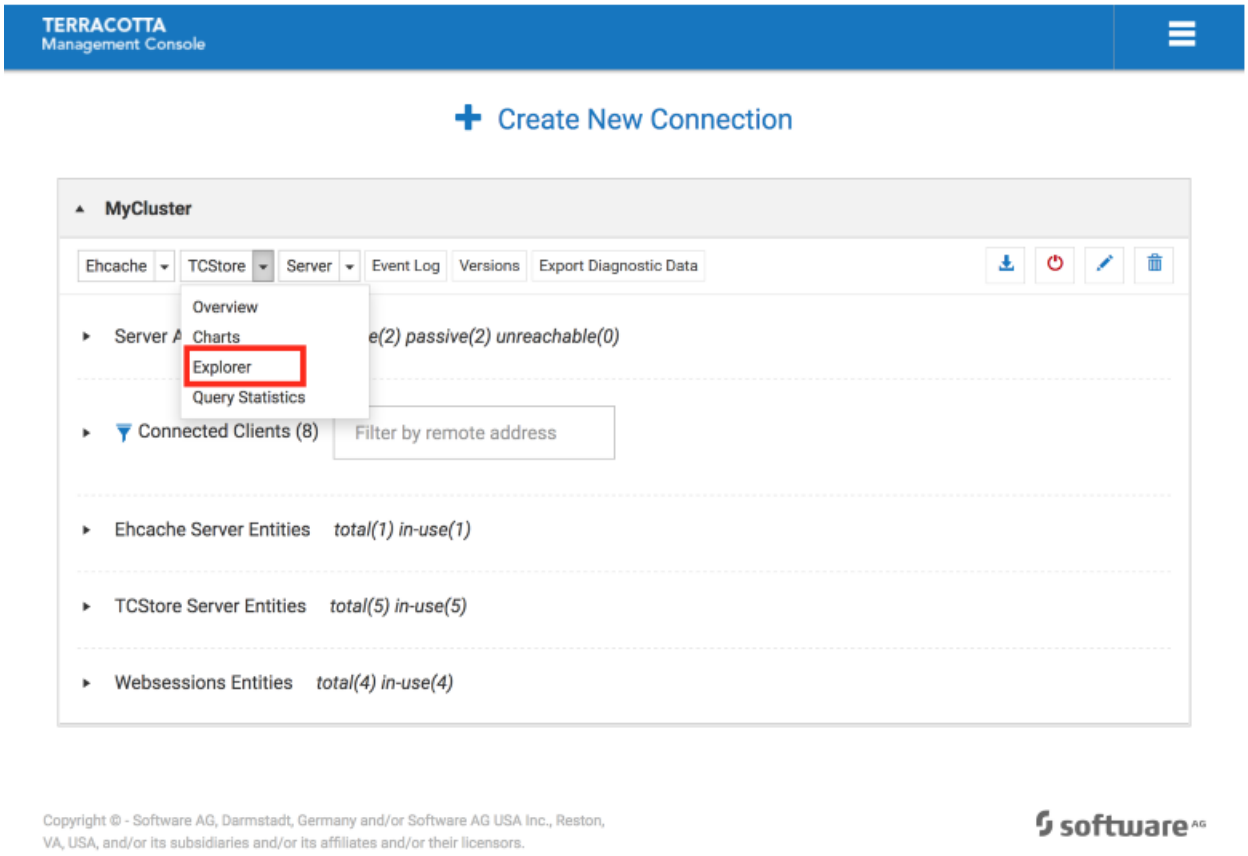
Querying clustered datasets create short-lived clients. These clients are tagged with `tmc-excluded` and thus won't appear in the TMC UI.

Access to the panel

From the Landing Page,

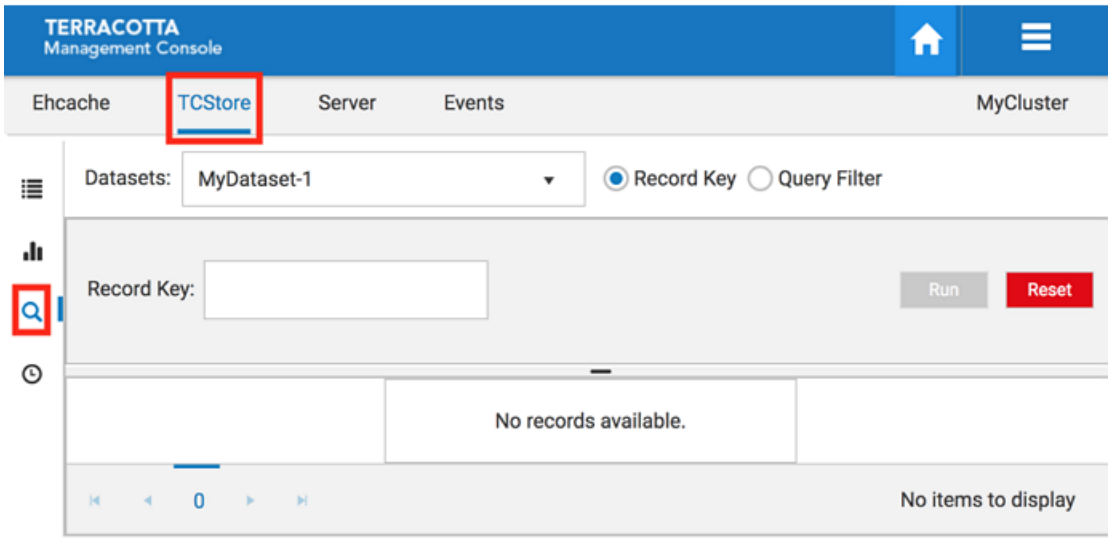
Select "Explorer" under "TCStore" button group,

Screenshot: Ad-hoc query from the landing page using the TCStore Explorer.



You will see the Explorer Panel (under the TCStore tab)

Screenshot: Explorer Panel, Search icon highlighted.



A dataset must be a selected, as every query is executed against a particular dataset. If your cluster already contains datasets, the dropdown list will be populated. From this dropdown the dataset to be queried can be selected. If there is no dataset existing in your cluster, there will be a notification window indicating the error.

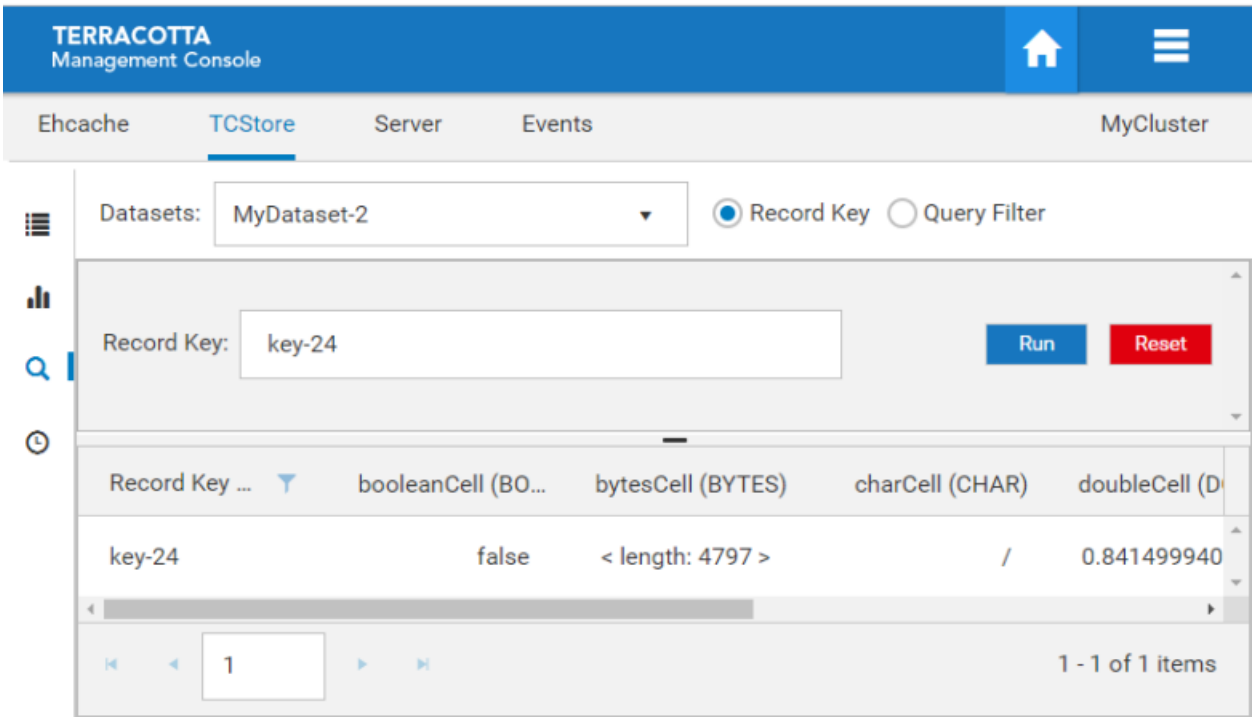
There are 2 query methods to choose from:

1. Query by Record Key
2. Query by Filters

Query by Record Key

If you know the record key you want, you can query by record key. Record key can be all valid Cell types except BYTES. Normally, the result should contain only one record.

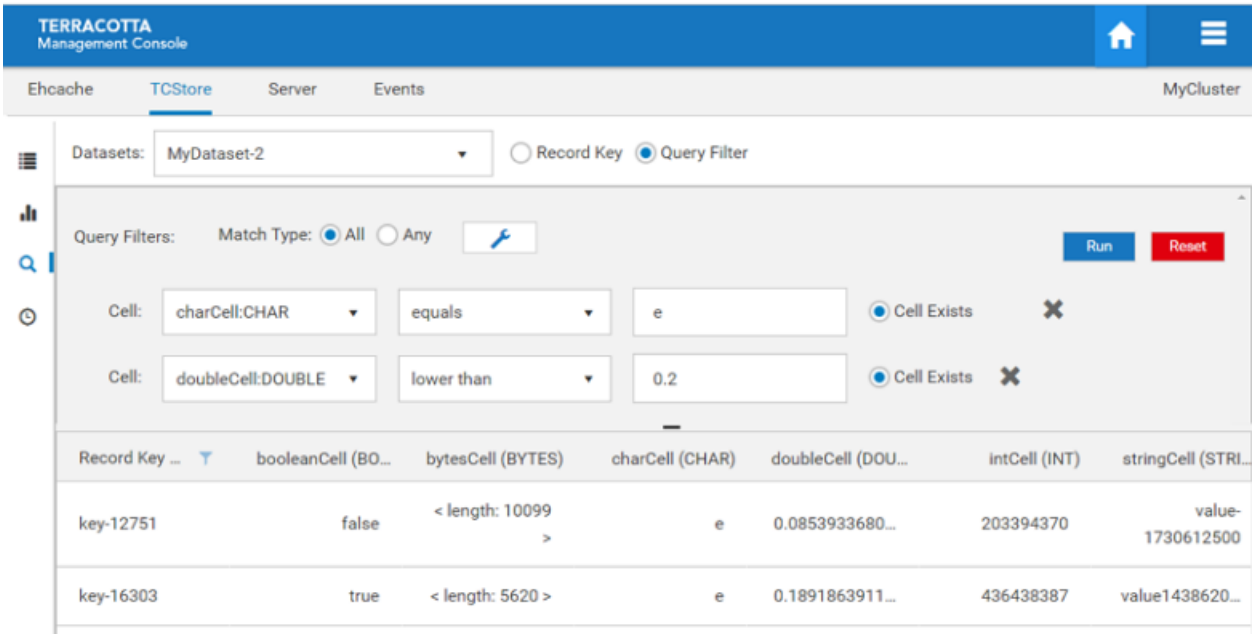
Screenshot: Explorer Panel, entering a Record key.



Query by Filters

If you don't know the record key, but you have some ideas about how certain Cells should look like, you can search by filters. The limitation is: You cannot filter by BYTES Cells.

Screenshot: Explorer Panel, search by filters.



You can add or remove filters. Each filter is targeted at a certain cell type (defined by cellName:cellType). Depending on the cell type, you have different query options. For example, if you select a STRING cell, then the operators can be "equals" and "starts with", which corresponds

to the portable TCStore API. Additionally, you can uncheck the "cell exists" option, and filter for all the records that MUST NOT contain the cell. This is different to not specifying any filter for the cell: If you do not specify any filter for "myStringCell", the returned records MAY OR MAY NOT contain "myStringCell". If any filter input field contains characters that cannot be converted to the target cell type, then an error notification will be displayed.

To simplify things, all the filters share the same connecting logic: You can choose "match All" or "match Any". Default is "match All".

When you think your filters look good, you can click "Run" button, and the query will be executed in the cluster against the selected dataset, in an optimized way. If you want to start over, click "Reset" button, it will clear all filters.

Grid Details

Screenshot of Grid Details

Record Key (LONG)	myBoolCell (BOOL)	myBytesCell (BYTES)	myStringCell (STRING)	myLongCell (LONG)
743	true	< length: 7 >	f5760aa	N/A
167	true	< length: 2 >	fd	N/A
436	true	< length: 3 >	f6fc95	N/A
2259	true	< length: 8 >	f6d9f27	N/A

1 - 4 of 4 items

The first column will always be Record Keys, with the key type shown in the header. Each cell header will also include the cell type. If the cell type is BYTES, the content is not rendered in String, as this would be meaningless for human consumption. Instead, the length of the Bytes content is displayed. If a record does not contain a certain cell, then you will see a specially formatted [underline]N/A. This is to visually differentiate from a String cell with content "N/A".

You can choose to sort records by column, by clicking any of the column headers. You can further filter records by records keys, by clicking on the "funnel" icon in Records Key column header.

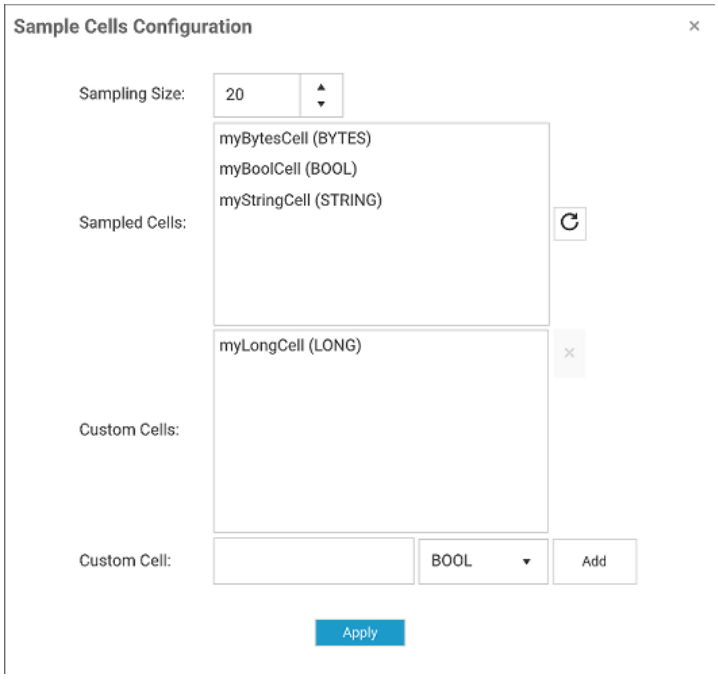
The maximum number of records that can be returned is set to 100, a pre-determined value, intended to avoid the complete dataset to be returned to the client (and front-end UI). It is recommended to specify sufficient filters to narrow the return to only a couple of records.

Sample Cells and its configuration (advanced topic)

If you click on Sample Cells configuration button,

you will see the Sample Cell Configuration popup window,

Screenshot: Sample Cell Configuration



Sample Cells is an advanced feature. Use it only if you are sure that the returned records do not match your expected schema. Since TCStore datasets do not enforce any schema, you can have records with very different Cells (cell name and/or cell type). The management console has to send a query to sample some records/cells in order to get a general sense of the schema in order to parse and show results in the grid. The sampled cells may not represent the accurate schema. For example, you may have a cell called "myLongCell" but it's not sampled, so it will not show up in the grid. In such a situation, you have 2 options:

1. Re-sample cells until the schema matches
2. Add a custom cell with cell name "myLongCell" and cell type "LONG"

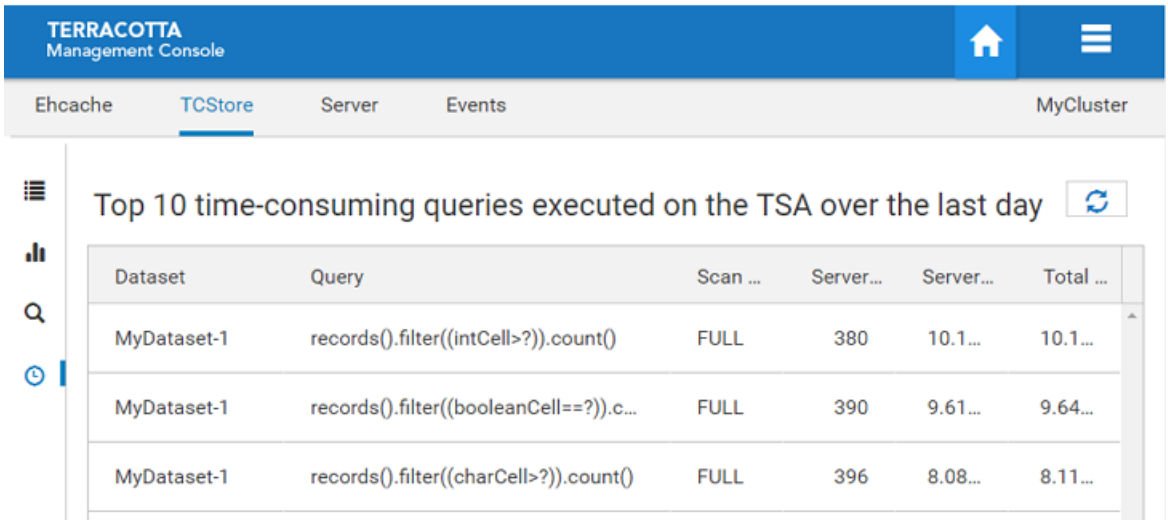
This UI window provides all features mentioned above. It features three major parts:

- You can specify a sample size. The larger the number, the more accurate, but the slower its performance. Default value is 20
- You can re-sample the cells and the results will be listed in "Sampled Cells" list
- You can add custom cells by specifying a cell name and cell type, and the current list of custom cells will be listed in "Custom Cells" list

Duplicated cells do not cause any problems, but we recommend to remove the duplicate cells in "Custom Cell" list and keep only what is needed.

The TCStore Query Statistics Panel

Screenshot: Query Statistics tab under TCStore.



The **Query Statistics** tab of the **TCStore** panel displays the top TCStore server-side query pipelines ordered by the total amount of server time they are costing.

Each query pipeline entry is comprised of the following fields:

Field	Description
Dataset	Name of the Dataset upon which the query was executed
Query	Shape of the server-side query pipeline
Scan Type	If the query pipeline took advantage of indexes (INDEXED) or needed to do a full scan (FULL)
Server Executions	Number of times the pipeline was presented to a server for execution
Server Time	Time taken (wall clock) by the servers to handle the query pipeline
Total Time	Time (wall clock) the query stream was held open

This information is not held by the TMS persistently, rather, it is obtained by querying each server.

These query statistics are refreshed automatically each time the tab is visited. The **Refresh** button at the top-left of the presentation is used to update the grid.

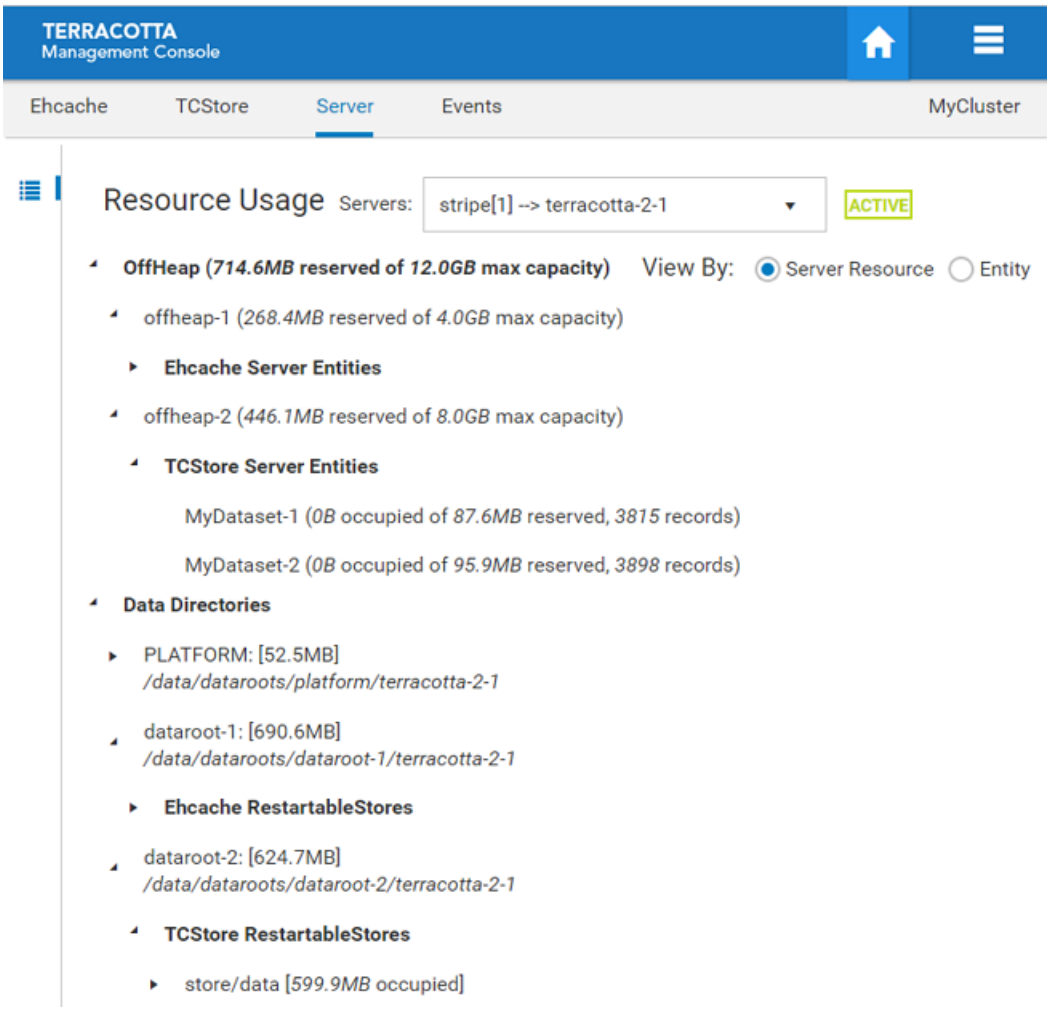
5 Using the Server Tab

The **Server** tab is where you can find information relating to the server-side of your application. There is currently a single panel, Resource Usage, described below.

Resource Usage Panel

The **Resource Usage** Panel displays information relating to your cache's or dataset's use of the Terracotta Server's configured resources, including *OffHeap* memory and Fast Restartable Store (*FRS*) data directories.

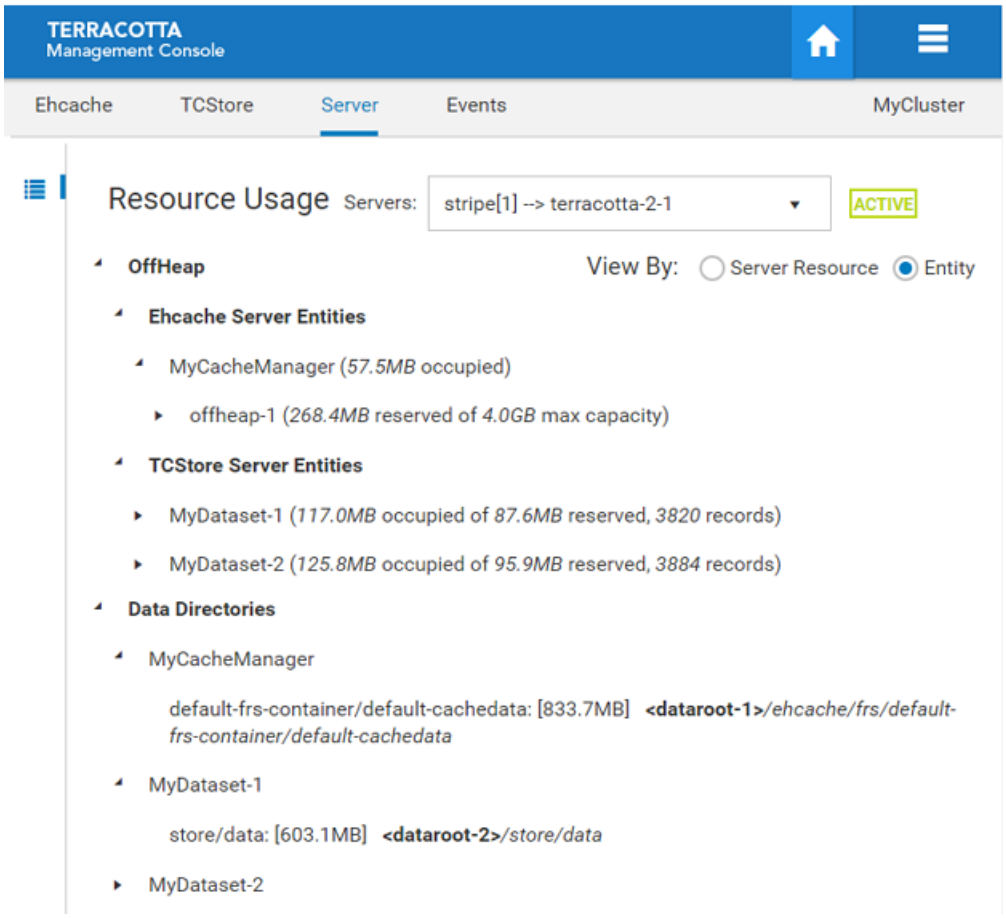
Screenshot: Resource Usage Panel



Use the **Servers** dropdown to view the resource usage of a particular Terracotta Server.

Use the **View By** radio buttons to toggle between a server resource or a caching entity-focused presentation, as shown below.

Screenshot: Resource Usage Panel, View By radio buttons



OffHeap

The **Server Resource** view shows how caching entities are making use of the allocated OffHeap resources configured for the server. Any number of server entities can use a particular server resource simultaneously.

The **Entity** view shows how caching entities are using the OffHeap resources configured for the server.

A caching entity can be configured to make use of one or more server resources via the pools (both shared and dedicated) they carve out of those server resources. Caches can be configured to store their entries in the pools configured by their containing CacheManager. The amount of space occupied by a caching entity is displayed, along with the total amount of space reserved for the pool.

While caching entities are associated with offheap resource pools, dataset entities make use of server offheap resources directly. The amount of space occupied by the dataset entity, the total amount of space reserved for the offheap server resource, as well as the count of records stored are displayed. Further, the cell names of the indexes that have been defined on your dataset, the type of index, the amount of offheap space occupied, the count of records indexed, and the number of times the index has been used are all displayed.

Data Directories

Data directories are configured in the server configuration and your application points to these named disk areas for persistent storage.

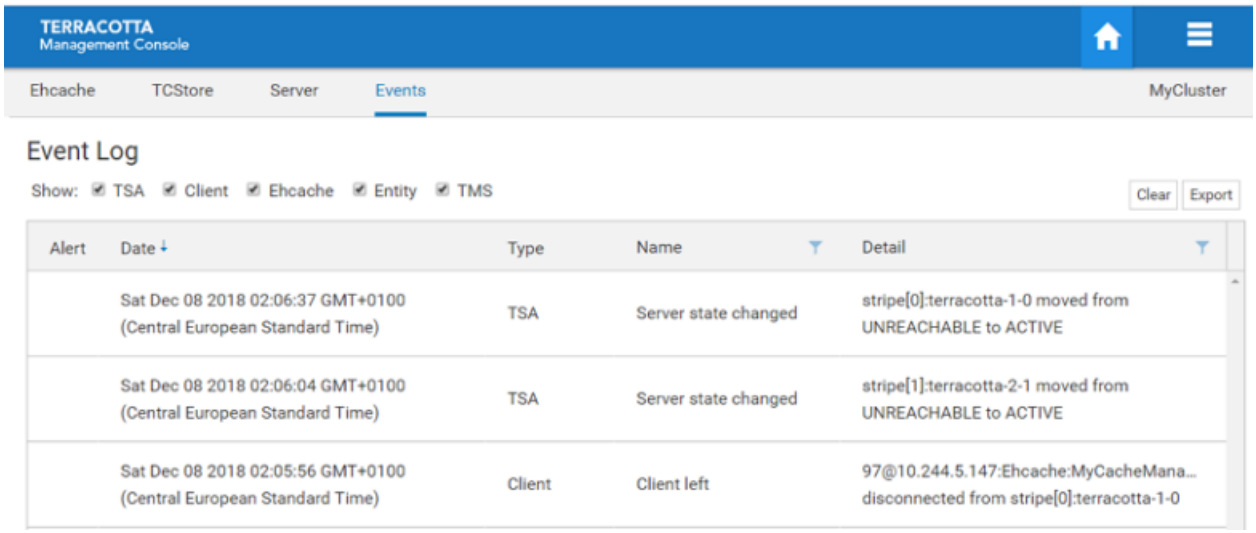
For each defined data directory, the filesystem location and occupied space are displayed, as well as the names of the server entities utilizing the directory and amount of filesystem space each is occupying.

6 Using the Events Tab

The **Events** tab displays the contents of the event log, in reverse chronological order. The event log is a record of important activities occurring in your cluster. The TMS listens for and persists these events, meaning that if the TMS is not running no events will be stored.

Event Log

Screenshot: TMC Event Log



The types of activities that generate events include:

- server state transitions
- clients connecting/disconnecting to TSA stripes
- server entities being created/destroyed
- resource constraint alerts
- and more

The event log is included in the set of diagnostics artifacts that can be downloaded from the cluster's connection panel on the Home Page, but in addition it can be downloaded separately using the **Export** button.

Events that are deemed to be critical are noted as alerts, displayed with an attention-focusing icon in the alert grid. Further, alert events are displayed in a temporary popup and the count of un-read alerts is shown in the cluster's connection panel on the Home Page.

Newly added events are shown in the grid with bold text, and reset to normal after navigating away.

An event is comprised of the following fields:

alert	Is this event deemed critical?
Timestamp	Time at which the TMS recorded the event
Type	Categorization of the event
Name	Sub-categorization
Detail	Description of the event

The maximum number of events that are saved in the TMS is controlled by the configuration property `tms.eventLogMaxRecords`, whose default value is 5000. The event log store in the TMS, as well as the event log grid, can be cleared using the **Clear** button.

7 Prometheus Integration

https://prometheus.io/ is an open-source systems monitoring and alerting toolkit which can be used alongside products like https://grafana.com/ for interactive data visualization and analytics. Terracotta provides a list of key Terracotta metrics in Prometheus compatible format over HTTP on the TMS (Terracotta Management Server) endpoint:

```
http(s)://[host]:[port]/actuator/prometheus
```

For example, if the Terracotta Management Server (TMS) is available at `http://localhost:9480`, and you have configured a cluster connection within its interface, then the prometheus metrics can be accessed at `http://localhost:9480/actuator/prometheus`.

Available metrics

All the available Terracotta metrics are prefixed with `sag_tc_`

Server Side Metrics

These are the same metrics as you would find under in the "Using the Server Tab" on page 51 section.

Server Specific Resource Usage Metrics

Prometheus Metric Name	Metric Description	Type
sag_tc_server_dataroot_total_disk_usage_bytes	Dataroot total disk usage in bytes.	Gauge.
sag_tc_server_offheap_allocated_memory_bytes	Offheap memory allocated in bytes.	Gauge.
sag_tc_server_restartable_store_total_usage_bytes	FRS usage in bytes.	Gauge.

Server-side Caching Specific Resource Usage Metrics

Prometheus Metric Name	Metric Description	Type
sag_tc_server_caching_pool_allocated_size_bytes	Caching pool allocated size in bytes.	Gauge.

Prometheus Metric Name	Metric Description	Type
sag_tc_server_caching_store_allocated_memory_bytes	Caching store allocated memory in bytes.	Gauge.
sag_tc_server_caching_store_data_size_bytes	Caching store data size in bytes.	Gauge.
sag_tc_server_caching_store_entries_count	Number of Caching store entries.	Gauge.

Server-side Store Specific Resource Usage Metrics

Prometheus Metric Name	Metric Description	Type
sag_tc_server_dataset_main_record_occupied_storage_bytes	Total occupied storage by the dataset in bytes - this is the sum of the 3 `dataset_occupied` metrics below	Gauge.
sag_tc_server_dataset_occupied_primary_key_bytes		Gauge.
sag_tc_server_dataset_occupied_persistent_support_bytes		Gauge.
sag_tc_server_dataset_occupied_heap_bytes		Gauge.
sag_tc_server_dataset_allocated_memory_bytes	Total allocated storage by the dataset in bytes - this is the sum of the 4 dataset_allocated metrics below	Gauge.
sag_tc_server_dataset_allocated_primary_key_bytes		Gauge.
sag_tc_server_dataset_allocated_persistent_support_bytes		Gauge.
sag_tc_server_dataset_allocated_heap_bytes		Gauge.
sag_tc_server_dataset_allocated_index_bytes		Gauge.
sag_tc_server_dataset_index_access_count	Dataset index access count.	Counter.
sag_tc_server_dataset_index_occupied_storage_bytes	Dataset index occupied storage in bytes.	Gauge.
sag_tc_server_dataset_index_record_count	Dataset index record count.	Gauge.
sag_tc_server_dataset_record_count	Dataset record count.	Gauge.

All the exposed server metrics have the following labels:

Server Metric Label	Label Description
alias	This label can represent a server-side cache resource name, an offheap resource name, a dataroot name, or a dataset name.
connection_name	Name of the connection set by the user in TMC web application.
entity_name	A technical attribute that represents the server-side entity name.
entity_type	A technical attribute that represents the server-side entity type.
server	Represents the server name.
stripe	Represents the stripe name.
cluster_tier_manager	For caching resource only. Matches the alias of the entity given by the user when connecting to a clustered cache, e.g. - `terracotta://myhost:9410/anEntity`

Example

```
sag_tc_server_caching_pool_allocated_size_bytes{alias="cache1",
cluster_tier_manager="CacheManager1", connection_name="MyCluster",
entity_name="CacheManager1$cache1", entity_type="cache_cluster_tier",
instance="localhost:9480", job="terracotta", server="stripe-1-server-1",
stripe="stripe-1"} 2228224
```

This example represents server side cache store allocated size in bytes created by the cache "_cache1_" under cluster_tier_manager named "_CacheManager1_" in server named "_stripe-1-server-1_" for connection called "_MyCluster_".

Cache Metrics

These are the same metrics as you would find under the "Using the Ehcache Tab" on page 25 section.

Prometheus Metric Name	Metric Description	Type
sag_tc_cache_get_hit_latency_100_percentile	Latency maxima of successful cache.get(key) operations (hits).	Gauge.
sag_tc_cache_get_hit_latency_95_percentile	95th percentile of latencies of successful cache.get(key) operations (hits).	Gauge.

Prometheus Metric Name	Metric Description	Type
sag_tc_cache_get_hit_latency_99_percentile	99th percentile of latencies of successful cache.get(key) operations (hits).	Gauge.
sag_tc_cache_get_miss_latency_100_percentile	Latency maxima of misses of cache.get(key) operations.	Gauge.
sag_tc_cache_get_miss_latency_95_percentile	95th percentile of latencies of misses of cache.get(key) operations.	Gauge.
sag_tc_cache_get_miss_latency_99_percentile	99th percentile of latencies of misses of cache.get(key) operations.	Gauge.
sag_tc_cache_put_latency_100_percentile	Latency maxima of successful cache.put(key, val) operations.	Gauge.
sag_tc_cache_put_latency_95_percentile	95th percentile of latencies of successful cache.put(key, val) operations.	Gauge.
sag_tc_cache_put_latency_99_percentile	99th percentile of latencies of successful cache.put(key, val) operations.	Gauge.
sag_tc_cache_remove_latency_100_percentile	Latency maxima of successful cache.remove(key) operations.	Gauge.
sag_tc_cache_remove_latency_95_percentile	95th percentile of latencies of successful cache.remove(key) operations	Gauge.
sag_tc_cache_remove_latency_99_percentile	99th percentile of latencies of successful cache.remove(key) operations	Gauge.
sag_tc_cache_hit_count_total	Total times a get command returned a value.	Counter.
sag_tc_cache_miss_count_total	Total times a get command did not return a value.	Counter.
sag_tc_cache_put_count_total	Total number of puts to the cache.	Counter.
sag_tc_cache_removal_count_total	Total number of removes from the cache.	Counter.

Prometheus Metric Name	Metric Description	Type
sag_tc_clustered_hit_count_total	Total number of get commands that returned a value from the cluster tier.	Counter.
sag_tc_clustered_miss_count_total	Total number of get commands that failed to return a value from the cluster tier.	Counter.
sag_tc_clustered_put_count_total	Total number of puts to the cluster tier.	Counter.
sag_tc_clustered_removal_count_total	Total number of removes from the cluster tier.	Counter.

Cache Metric Label	Label Description
cache	Name of the cache.
cache_manager	Name of the cache manager.
client	Gives information about client. (e.g. 32164@127.0.0.1:Ehcache:CacheManager1)
client_address	Address part of client. (e.g. 127.0.0.1)
client_name	Name part of client. (e.g. Ehcache:CacheManager1)
client_pid	PID part of client (e.g. 32164)
connection_name	Name of the connection set by the user in TMC web application.
clustered	`Y` if cache is clustered, `N` if not clustered.
instance_id	Unique ID representing client.

Example

```
sag_tc_cache_get_hit_latency_95_percentile{cache="cache1",
cache_manager="CacheManager1",
client="32164@127.0.0.1:Ehcache:CacheManager1",
client_address="127.0.0.1", client_name="Ehcache:CacheManager1",
client_pid="32164", clustered="Y", connection_name="MyCluster",
instance="localhost:9480", instance_id="84bd0e20-26ff-4b9f-ae6c-90622eb48c74",
job="terracotta"} 2110940
```

Store Metrics

These are the same metrics as you would find under the "Using the TCStore Tab" on page 35 section.

Prometheus Metric Name	Metric Description	Type
sag_tc_dataset_add_latency_100_percentile	Latency maxima of dataset add operations.	Gauge.
sag_tc_dataset_add_latency_95_percentile	95th percentile of latencies of dataset add operations.	Gauge.
sag_tc_dataset_add_latency_99_percentile	99th percentile of latencies of dataset add operations.	Gauge.
sag_tc_dataset_delete_latency_100_percentile	Latency maxima of dataset delete operations.	Gauge.
sag_tc_dataset_delete_latency_95_percentile	95th percentile of latencies of dataset delete operations.	Gauge.
sag_tc_dataset_delete_latency_99_percentile	99th percentile of latencies of dataset delete operations.	Gauge.
sag_tc_dataset_get_latency_100_percentile	Latency maxima of dataset read operations.	Gauge.
sag_tc_dataset_get_latency_95_percentile	95th percentile of latencies of dataset read operations.	Gauge.
sag_tc_dataset_get_latency_99_percentile	99th percentile of latencies of dataset read operations.	Gauge.
sag_tc_dataset_update_latency_100_percentile	Latency maxima of dataset update operations.	Gauge.
sag_tc_dataset_update_latency_95_percentile	95th percentile of latencies of dataset update operations.	Gauge.
sag_tc_dataset_update_latency_99_percentile	99th percentile of latencies of dataset update operations.	Gauge.
sag_tc_dataset_add_already_exists_total	The number of Add:AlreadyExists operations.	Counter.
sag_tc_dataset_add_failure_total	The number of Add:Failure operations.	Counter.
sag_tc_dataset_add_success_total	The number of Add:Success operations.	Counter.
sag_tc_dataset_delete_failure_total	The number of Delete:Failure operations.	Counter.
sag_tc_dataset_delete_not_found_total	The number of Delete:NotFound operations.	Counter.

Prometheus Metric Name	Metric Description	Type
sag_tc_dataset_delete_success_total	The number of Delete:Success operations.	Counter.
sag_tc_dataset_get_failure_total	The number of Get:Failure operations.	Counter.
sag_tc_dataset_get_not_found_total	The number of Get:NotFound operations.	Counter.
sag_tc_dataset_get_success_total	The number of Get:Success operations.	Counter.
sag_tc_dataset_stream_failure_total	The number of Stream:Failure operations.	Counter.
sag_tc_dataset_stream_request_total	The number of Stream:Request operations.	Counter.
sag_tc_dataset_update_failure_total	The number of Update:Failure operations.	Counter.
sag_tc_dataset_update_not_found_total	The number of Update:NotFound operations.	Counter.
sag_tc_dataset_update_success_total	The number of Update:Success operations.	Counter.

Store Metric Label	Label Description
dataset	Name of the dataset.
dataset_manager	Name of the dataset manager.
dataset_instance	Name of the dataset instance.
client	Gives information about client. (e.g. `32164@127.0.0.1:Store:TinyPounderDataset`)
client_address	Address part of client. (e.g. `127.0.0.1`)
client_name	Name part of client. (e.g. `Store:TinyPounderDataset`)
client_pid	PID part of client (e.g. `32164`)
connection_name	Name of the connection set by the user in TMC web application.
instance_id	Unique ID associated with each client.

Example

```
sag_tc_dataset_add_latency_95_percentile{client="32164@127.0.0.1:Store:TinyPounderDataset",

client_address="127.0.0.1", client_name="Store:TinyPounderDataset", client_pid="32164",

connection_name="MyCluster", dataset="dataset1", dataset_instance="dataset1-1",
dataset_manager="TinyPounderDataset",
instance="localhost:9480", instance_id="1UnvihEwPjFnfjnvGO_MoA", job="terracotta"}
```

Connecting to Prometheus with Security disabled in TMS

Follow these steps to connect to the Prometheus endpoint with Security disabled in TMS:

1. Navigate to the TMC web application and create a connection to the TSA cluster.

   The name of the connection will be the value of the `connection_name` label. `/actuator/prometheus` will start returning terracotta metrics in prometheus format.

2. Use the following sample configuration to add Terracotta as a target in the prometheus.yml configuration file.

   For more details, refer to the https://prometheus.io/docs/prometheus/latest/configuration/configuration/ page.

   ```
   global:
     scrape_interval: 30s
   scrape_configs:
     - job_name: 'terracotta'
       metrics_path: /actuator/prometheus
       static_configs:
       - targets: ['localhost:9480']
   ```

Connecting to Prometheus with Security enabled in TMS

When user authentication is enabled, all endpoints on TMS become password protected. TMS supports basic authentication scheme to access the `/actuator/prometheus` endpoint. In order for Prometheus to access the metrics, you need to provide TMS credentials in the `basic_auth` key of the prometheus configuration file. Follow these steps to connect to the Prometheus endpoint with Security enabled in TMS:

1. Navigate to the TMC web application and create a connection to the TSA cluster.

   The name of the connection will be the value of the `connection_name` label.

2. If SSL is enabled, export the SSL certificate and provide it to the prometheus configuration file in a key called `ca_file`. To do that, you can use either a command line tool or a graphical tool, like https://keystore-explorer.org.

   For example:

   ```
   keytool -exportcert -alias <tms-alias> -keystore <tms-keystore> -rfc -file <tms-cert>
   ```

3. Use the sample configuration to add Terracotta as a target in the prometheus.yml configuration file.

<username> and *<password>* are the user's username and password, *<path-to-tms-certificate>* is the valid file path to the TMS certificate.

```
global:
  scrape_interval: 30s
scrape_configs:
  - job_name: 'terracotta'
    scheme: https
    metrics_path: /actuator/prometheus
    static_configs:
    - targets: ['localhost:9480']
    basic_auth:
      username: <username>
      password: <password>
    tls_config:
      ca_file: <path-to-tms-certificate>
```

Rates and Ratios

Terracotta Prometheus endpoint exposes counter metrics from which associated rates and ratios can be calculated with the help of https://prometheus.io/docs/prometheus/latest/querying/basics/. For example:

- You can derive cache hit rate from the `sag_tc_cache_hit_count_total` counter by using the PromQL rate function.

  ```
  rate(sag_tc_cache_hit_count_total[2m])
  ```

  This will calculate the average cache hit rate (hits/second) measured over a 2 minutes window.

- You can calculate the cache hit ratio with the following query (range = 2 minutes).

  ```
  (rate(sag_tc_cache_hit_count_total[2m]) /
  (rate(sag_tc_cache_hit_count_total[2m]) +
  rate(sag_tc_cache_miss_count_total[2m])))
  ```

Note:
Collector Intervalcontrols how frequently the statistics will be collected in TMS. To achieve graph trends similar to TMC, you can set `scrape_interval` in the prometheus.yml to be equal to collector interval. The default value of collector interval is 30 seconds. The range value in the rate function should be changed based on the value of `scrape_interval`.

Getting the metrics directly from the servlet

Using any HTTP client, such as curl :

```
curl http(s)://[host]:[port]/actuator/prometheus
```

You can craft a regular expression to search for specific metrics :

```
curl -s http(s)://[host]:[port]/actuator/prometheus |
grep "sag_tc_server.*{.*} .*"
```

Prometheus in Kubernetes

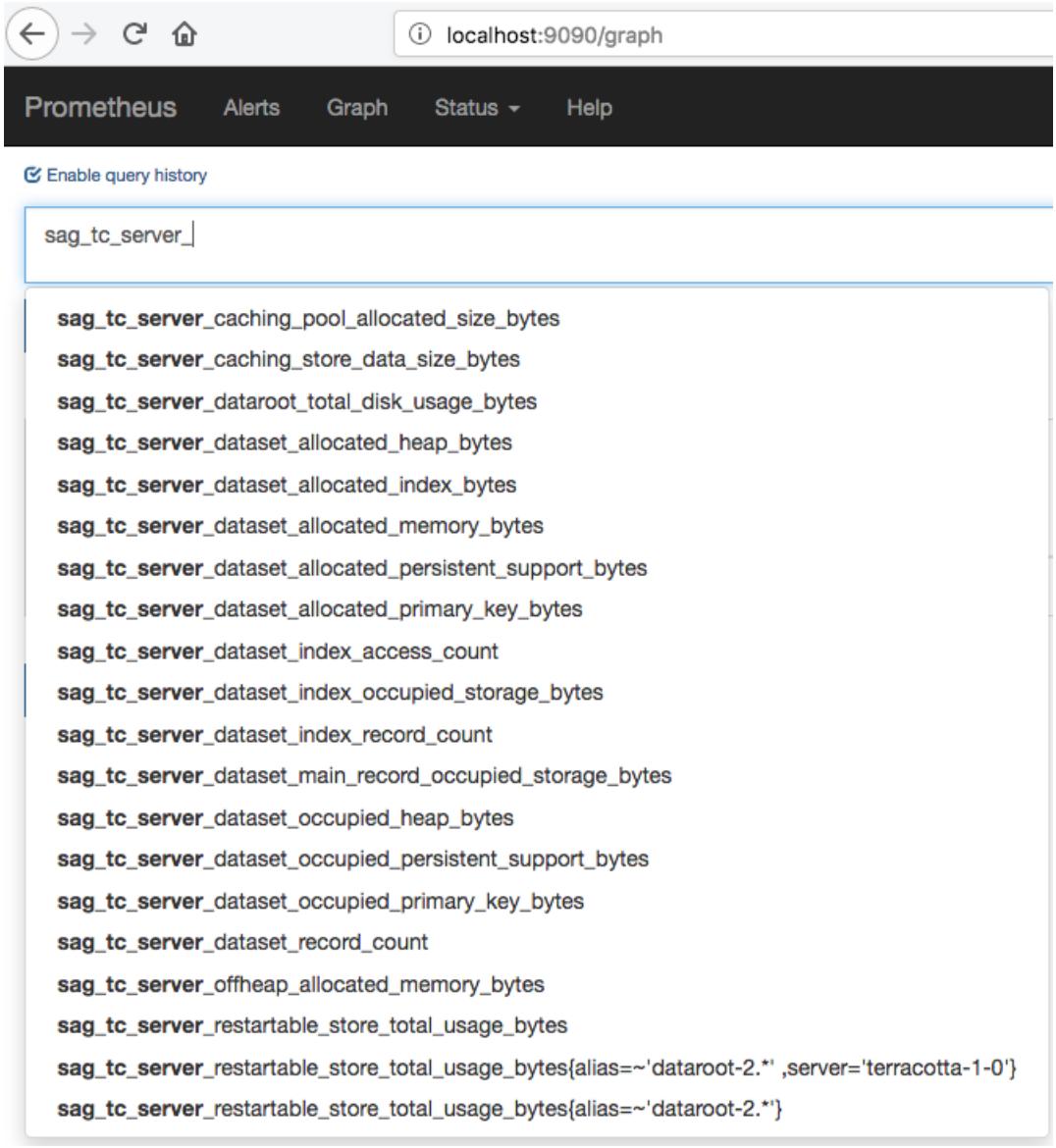
There are several ways to let Prometheus grab the metrics available at `http(s)://[host]:[port]/actuator/prometheus`.

For example, if you deployed the TMC using Kubernetes, and you created a service for it, you can simply add this YAML configuration to your service manifest to have Prometheus read the metrics regularly:

```
metadata:
  name: tmc
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/path: '/actuator/prometheus'
```

Prometheus querying

When you have successfully installed and deployed Prometheus, you'll be able to choose the Terracotta cluster metrics.



You can also precisely choose which metrics you want to display using the labels and PromQL. For example, if you only want the FRS usage for the "dataroot-2" dataroot on the server "terracotta-1-0", you can use the following Prometheus query :

```
sag_tc_server_restartable_store_total_usage_bytes{
alias=~'dataroot-2.*',server='terracotta-1-0'}
```

Visualization with Grafana

Once you have deployed Prometheus, you can use Grafana for data visualizations and monitoring.

To get started, you can import a sample dashboard for Terracotta from the here.

Prometheus Usage Notes

- If you remove a cluster connection from TMC, the Prometheus endpoint will clear all the Terracotta metrics for that connection name.
- If the Prometheus server is unable to get metrics of Terracotta, make sure that the cluster is successfully connected in the TMC web UI dashboard. For additional details, check the TMC and Prometheus logs.

8 Performance Considerations

The TMC application being both a statistics database and an application server on the same process, we strongly advise you to allocate at least 2GB of offheap memory (`tms.offheapSizeMB=2048`). Also, this memory amount will depend on how many statistics are stored and whether the persistence mode is `INMEMORY` or `HYBRID` (see below).

The amount of statistics the TMC can store depends on the number of objects in your cluster (not the number of clients). The more objects (caches or datasets) in your clients, the more memory/disk the TMC internal datastore will require.

For example, with the following configuration and a collector interval set to 30 seconds, and a cluster of 60 objects (mix of datasets and caches), the TMC's offheap memory will typically grow about 180MB per hour.

```
tms.offheapSizeMB=2048
tms.persistenceMode=INMEMORY
```

There are several parameters you can control depending on your cluster size and the quantity of history you want to keep.

- `Collector Interval`: In TMC, this controls how frequently the statistics will be collected and stored in the TMC database.
- `tms.persistenceMode`: `INMEMORY` will keep all the data in offheap memory (and disk). This speeds up the queries but requires a lot of memory if you want to keep a large history. `HYBRID` will keep only the indexes in memory. The query speed will depend on the disk speed.
- `tms.offheapSizeMB`: This controls the maximum offheap memory allowed for the TMC. `HYBRID` requires less memory than `INMEMORY`.
- `tms.offheapThreshold`: This is the threshold (as a percentage of `tms.offheapSizeMB`) above which the TMC will remove the oldest statistics from the database. This purging process is required so that the TMC does not run out of memory. 80% is a good value for 2GB of offheap memory when using `INMEMORY`.
- `tms.statisticsMaxAgeMinutes`: This parameter controls the maximum time (in minutes) the statistics will remain in the database. Statistics stored before this delay get removed. This parameter has an effect on the time frames you will be able to see in the TMC UI, assuming you have enough memory and/or disk space to store the statistics for this period of time.

9 Migrating the Terracotta Management Console

If you install a new 10.3 Terracotta Management Console over an existing installation of a previous 10.2 Terracotta Management Console , you can expect a data migration to happen.

This data migration will preserve your connections, so that you won't need to re-create them in your new 10.3 Terracotta Management Console instance.

There is no further interaction needed, except to ensure that the property

```
tms.storageFolder
```

still points to the same folder as your previous 10.2 installation did.

When starting the 10.3 Terracotta Management Console for the first time, the migration process will produce a log entry similar to the following example:

```
WARN  c.t.m.s.migration.MigrationService - Backing up data before migration
WARN  c.t.m.s.migration.MigrationService - Done backing up data to
  /opt/softwareag/management-console/data-10.2-for-test.backup-1534351786349
WARN  c.t.m.s.migration.MigrationService - Migrating data from 10.2 to 10.3
WARN  c.t.m.s.migration.MigrationService - Finished migrating data from 10.2 to 10.3
```

This indicates that the migration process was successful.