CA Test Data Manager (TDM)

Best Practice Guide and Health Check

Version 2.2

Version:	2.3
Date:	Tuesday, 21st January 2020
Authors:	Keith Puzey, Abderrahmane Zahrir, Walter Guerrero, Salvator Pilo

Referenced Documents

Related Project Documentation

CA TDM Product Documentation Bookshelf Broadcom Techdocs

Contents

Chapter 1: Foundation Functional Feature

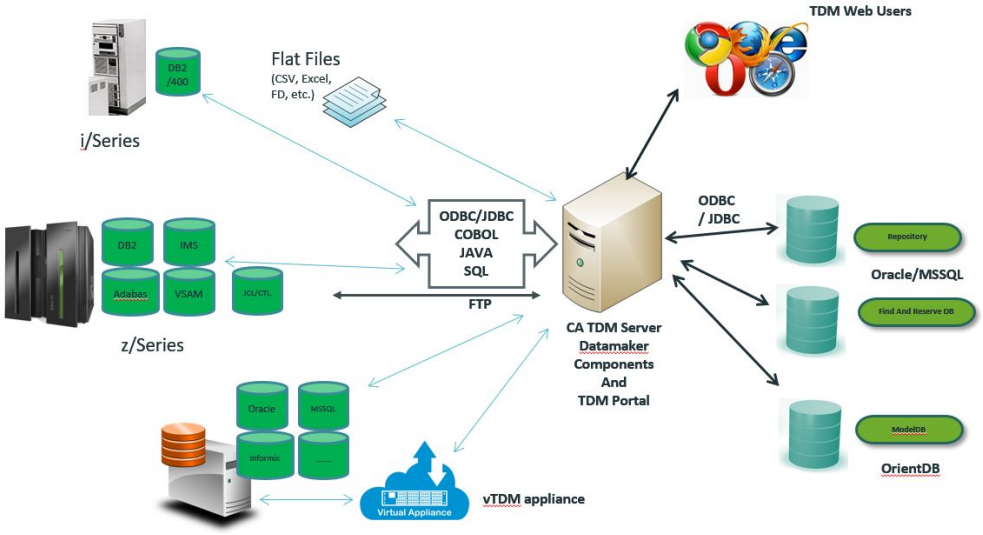
Planned End State

The planned state is to provide a reference architecture to support the implementation of CA TDM.

This document can be considered as the logical and physical design, illustrating how the technical solution will be implemented to meet the architecture (non-functional and environment constraints) requirements and how it will be configured or customized to support the requirements.

All content contained in this document is based on CA Lead Practices and where necessary, it has been updated to reflect the corporate governance standards for architecture requirements.

The following shows a basic TDM logical system architecture. The core components of CA Test Data Manager are normally deployed on a Windows server. The TDM Portal is a web based UI which can be used to interact with TDM. TDM Portal can be deployed on windows as well as in a Docker container (Ubuntu) . The datamaker component is a legacy component of TDM which is required for some of the TDM functions and administration. A full list of the TDM components can be found here



Solution Personas

The following table summarizes the solution personas and how they can use the CA TDM solution. Where applicable, the personas are grouped by the capability in which they play a role:

Description	Persona (Role)
TDM Administrator	Configure CA TDM and administer Projects and users access permissions
Test Data Engineer (TDE)	Administer Data sources and create and manage: Data Generation rules and Jobs Find and Reserve Models Data Masking rules and Jobs
Data Compliance Auditor	Review PII Discovery models and create Audit Report
Data Compliance Approver	Review and Approve PII Audit reports
End User (Tester)	Use self-service catalogue to request test data and DB clones

Foundation Logical Architecture

The following architecture illustrates the required application component packaging for the CA TDM foundation solution.

Foundation System Specification Requirements

The hardware requirements for the solution, **for average capacity**, are defined in the following subsection.

Foundation Physical Architecture

The hardware required for the Foundation Physical Architecture solution is given in the following table.

TDM Component	Qty	Disk Space	TDM Repository Database
Database	1	50 GB	Microsoft SQL Server 2012, 2014, 2016, 2017 Microsoft SQL Server Express 2014, 2016, 2017 Oracle 11g and 12c, 18c Oracle 11g XE

TDM Component	Qty	RAM	CPUs	OS &SW Disk Space	Operating Systems
TDM Server (Medium Size)	1	16 GB	4 Cores, 2.5 Ghz	50 GB	Microsoft Windows Server 2008 R2 Microsoft Windows Server 2012 R2 (64-bit) Microsoft Windows 10 64-bit Microsoft Windows 7 64-bit Ubuntu 16.04 - Docker Deployable containers

Base System Configuration Requirements

Solution Component Ports

From	To	Port
ARD	TDM	TCP 8090 or Portal port 8080 / 8443
TDM Portal Data Maker Service Layer – TDOD Service	Repository	JDBC MSSQL 1433 /Oracle 1521 ODBC MSSQL 1433 /Oracle 1521 ODBC MSSQL 1433 /Oracle 1521
TDM Portal Datamaker	Active Directory	LDAP 389 / LDAPS 636 LDAP 389 / LDAPS 636
TDM Portal Remote Publish Service	Email Server	SMTP – TCP 25 SMTP – TCP 25
TDM Portal	OrientDB	TCP 2424
TDM Portal	Find and Reserve Database	JDBC MSSQL 1433 /Oracle 1521 (Location - TDM Repository as the default configuration)
Data Maker	Mainframe	FTP – TCP 22
TDM Portal	Masking Messaging Container	TCP 5671
Masking Container	Masking Messaging Container	TCP 5671

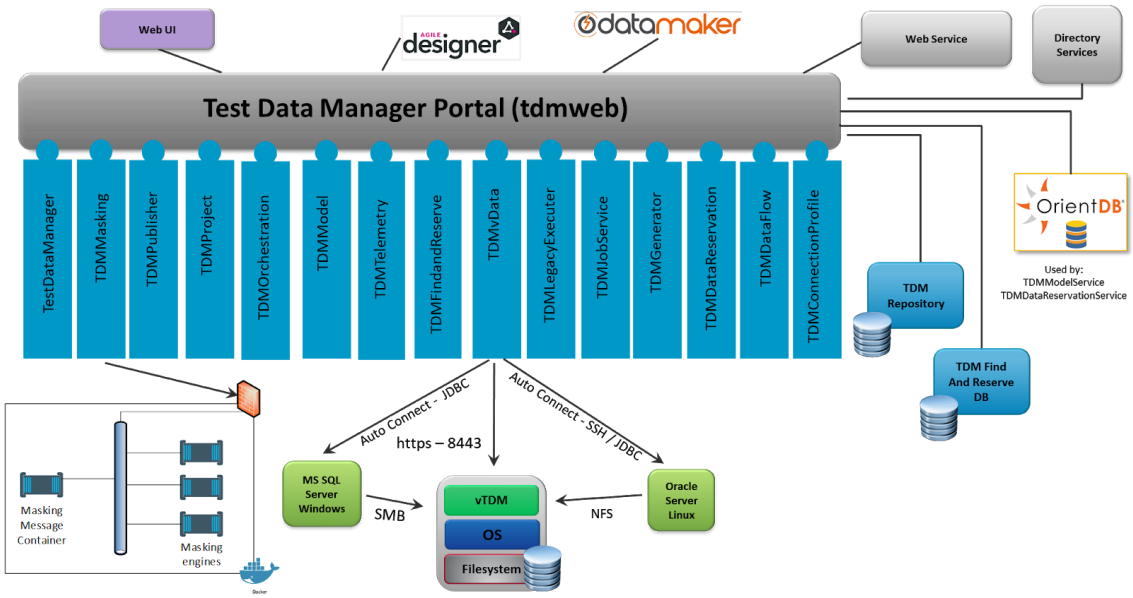
The following table summarizes the ports used by the Solution components:

Source and Target Database Ports

TDM Component	Database	Default Port
TDM Portal DataMaker	Oracle	TCP 1521
TDM Portal DataMaker	MS SQL	TCP 1433
TDM Portal DataMaker	IBM DB2 AS400	TCP 446
TDM Portal DataMaker	IBM DB2 z/OS	TCP 446
TDM Portal DataMaker	Teradata	TCP 1025
TDM Portal DataMaker	PostGres	TCP 5432
TDM Portal DataMaker	MySQL	TCP 3306

TDM Portal Architecture

The TDM Portal is based on Apache Tomcat and the TDM components are deployed as microservices. The following diagram shows the Microservices and a high level connectivity architecture.

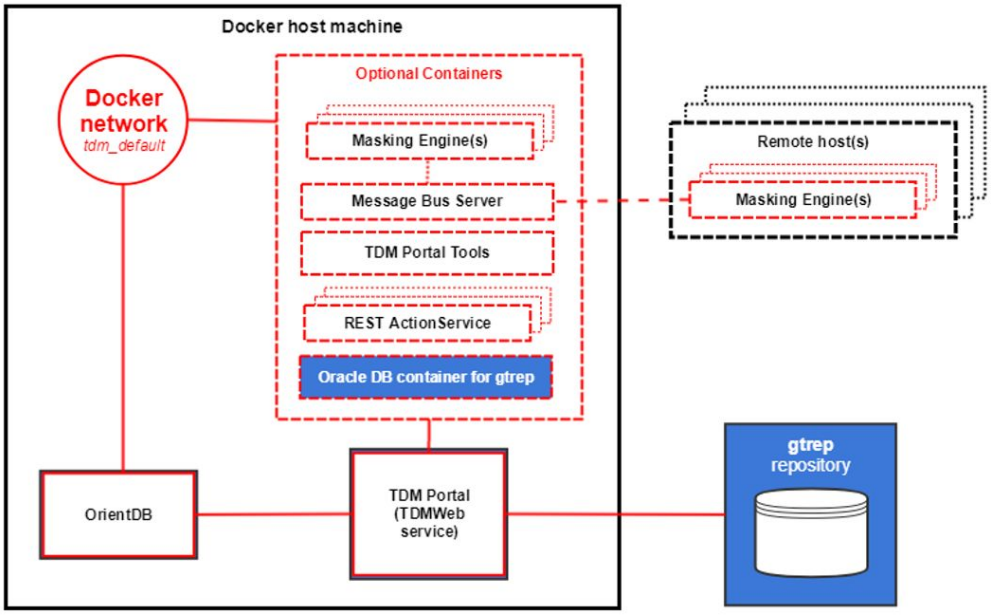


Additional Deployment method - Docker

A standard TDM deployment is normally on Windows but a subset of TDM can be deployed using the preshipped Docker images. These images are built on Ubuntu 16.04 and Java 1.8 v212.

Note: Certain features of TDM Portal are not available when using the docker containers.

This diagram shows how the Docker containers interact with each other.
When deploying TDM Portal using the Docker containers the only supported repository is Oracle, this oracle Database can either be within another Docker container or standalone.



The following list shows the shipped Docker images and their uses

Image Name	Description
Tdmweb:version	**TDM Web Portal**
Orientdb:version	**Orient Database**
tdmtools:version	**TDM Tools** Use this container to: ● Encrypt a password, Generate a JWT shared secret ● Create a gtrep user on your Oracle gtrep database ● Create sample databases on your Oracle database, from the sample databases supplied with CA TDM.

action-service:version	**Action Services** Use instances of this container to allow you to execute Actions from TDM Portal in Docker (one Action per container)
messaging:version	**Message Bus Server container** The TDM Portal service (in Windows or Docker) sends masking jobs (split into tasks) to this container, which then distributes these tasks to masking engine containers.
masking:version	**Masking engine container** This container performs masking tasks with the Fast Data Masker engine.

Chapter 2 :Masking Performance Optimization

CA TDM Portal 4.8 performs masking utilising CA Fast Data Masker. CA TDM Portal can run multiple instances of FDM concurrently (the maximum and default number of instances is 4). You may be able to perform your masking job faster, if you can split the job into smaller jobs that Portal can process with concurrent instances of FDM.

Assess the size of your environment

The size of the data set that you want to mask (i.e. number of tables in each database/schema, number of columns and rows in tables) has an effect on how much memory FDM needs to mask the data, and how long it takes. The amount of memory Fast Data Masker requires for a masking job generally increases linearly in relation to the number of tables, columns and rows to mask.

Tip: You can use the PARALLEL option on the Masking Settings to set the number of parallel Java threads. Within an instance of FDM, FDM creates a Java thread for each table.

Memory use in FDM

For every 1 million rows and 100 columns to mask, 1GB of memory is generally sufficient to maintain optimum performance (see table below).

Rows	Columns	Memory Recommended
1 million	100	1GB
2 million	100	2GB
2 million	200	4GB

Tip: The **HEAPSIZE** option on the Masking Settings page controls how much memory, in MB, TDM Portal assigns to each FDM instance. The default value is 1000MB (1GB) The allocation of less memory than this for each instance is likely to result in slower masking.

Optimize concurrent jobs in CA TDM Portal

CA TDM Portal will split a masking job into separate instances for each **Connection Profile**. A masking job can run on either of the following and environment which consist of data sources that it accesses through **Connection Profiles** or a specified **Connection Profiles**.

The creation of multiple Environments, each with one Connection Profile, **does not** improve masking efficiency. However, a Connection Profile can contain multiple schemas.

A masking job task is created for each set of tables with under one million rows within the same schema. This means that we will be concurrently running a masking job for these sets of tables linked to a particular schema.

The masking Service will also create a separate masking job task for each large table (a large table is defined by the number of rows and that number is configured through application.properties).

Note that by default a masking container can run a maximum of 4 FDM instances (with one schema on each instance).

Example 1: suppose we have a connection profile containing 10 schemas and we are masking all tables in every schema and we have one masking container. CA TDM creates 10 masking job tasks. One task for each set of tables in a specific schema. The first 4 masking job tasks will be processed by 4 FDM instances and the remaining 6 masking job tasks will be queued.

Example 2: Suppose we have a connection profile containing 10 schemas and we are masking all tables in every schema and we have one masking container. Moreover 2 tables are considered LARGE.

The masking service will therefore create 12 masking job tasks; 10 tasks for the 10 schemas and 2 more tasks for the 2 large tables.

Memory Usage for concurrent masking instances

The total memory required for a masking job that contains multiple concurrent instances of FDM is equal to the sum of the memory required for each instance of FDM.

For example, if your job contains **4 instances**, and each one requires **1GB** of memory, the total memory you need is **4GB**.

Results of insufficient memory

CA TDM Portal runs FDM on the same physical system, therefore for optimum masking performance, this system must have enough physical memory for all concurrent instances of FDM. Less memory can result in:

Slower performance of masking jobs

Slower CA TDM Portal performance

Maximise memory availability

To maximize memory available to CA TDM Portal, we recommend that you schedule jobs to run at a time when memory load is lower.

Split Tables

From TDM 4.8.135 and above, when masking a very large table with a primary key or a unique index, you can improve performance by using the following options.

LARGETABLESPLITENABLED

Enables large tables processing.

Set this parameter to Y to enable, and to N to disable.

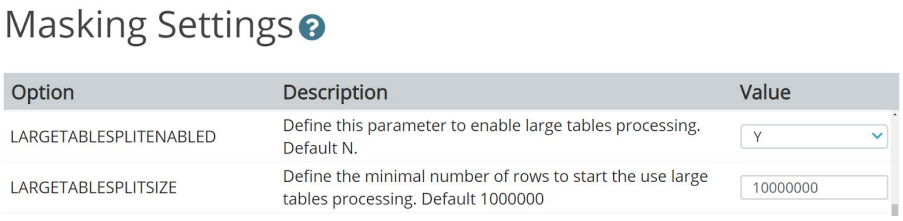
Default: N

LARGETABLESPLITSIZE

Defines the minimal number of rows for Fast Data Masker to start using large table processing.

Default: 1000000

With this setting, Fast Data Masker processes large tables by generating several blocks, with each block containing *LARGETABLESPLITSIZE* rows to be processed.



The existing option **PARALLEL** defines the number of threads that can run concurrently to process the blocks. If the **PARALLEL** option is not set, and you enable **LARGETABLESPLITENABLED**, then **PARALLEL** is set to 10 by default. If there are more blocks than threads, then remaining blocks are queued for processing and wait for a thread to become available.

The Parallel option is used to define the maximum number of separate threads that the FDM instance will utilise, each thread will consume additional CPU cycles as can be seen from the table in the masking example section.

Use Parallel Threads to Mask Data

Fast Data Masker lets you use parallel threads to mask large tables. The PARALLEL option enables you to run *n* concurrent threads.

To apply parallel threads, Fast Data Masker must split the work to be done into separate chunks. Fast Data Masker can manage this task in one of the following ways:

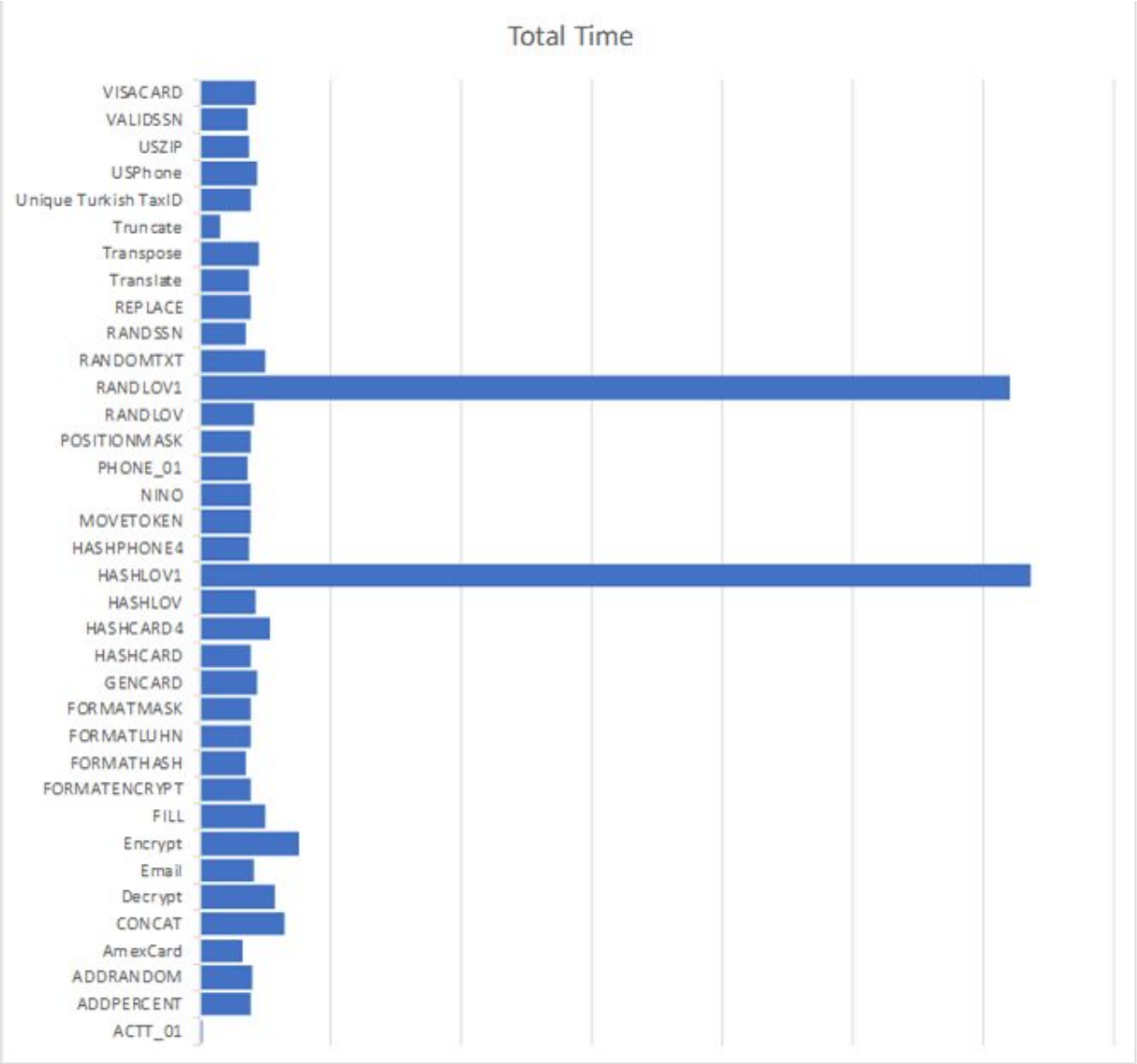
- A regular mask of multiple tables where none of the tables has a large amount of data. Fast Data Masker automatically sets a thread for each table.
- A large table that is not partitioned. In this case, split the table using the *where* clauses. For this to work, the following must apply:
    - The masking CSV must only contain a mask for a single table.
    - The *where* clauses must not overlap; for example, if two or more SQL Where clauses select the same rows in the table to be masked, then the mask causes row lock errors.
- For an Oracle-partitioned table, Fast Data Masker automatically assigns a thread to each underlying partition. This cannot be combined with the *where* clauses, and as in point two above, it is applicable to a mask for a single large table.

Note: The number of parallel threads that you can execute concurrently is constrained by the number of physical cores and/or processors available. If the parallel number specified in the options is greater than the number of cores, then some of the threads are held in a queue until resources become available.

You have successfully used parallel threads to mask the data.

Masking Function and Seed Lists

When reviewing the masking performance consideration should be made to the masking functions being used and confirming the optimum masking function is being used. Some masking functions require more resources and also will take longer to run as can be seen in the following graph. Broadcom engineering is currently working on updates to optimise these various masking methods.



As well as reviewing the masking functions being used consideration should be made to the size of any seed lists. Very large seed lists will slow down masking jobs

Run Fast Data Masker Scripts Remotely

When you save the defined masking information in Fast Data Masker, Fast Data Masker creates a batch file. This batch file includes information about the location of the connection file, masking file, options file, and other related information (for example, start memory). You can use this batch file to run from a remote location where Fast Data Masker is not installed. This is helpful in scenarios where you are facing performance issues on your server because of different components installed on the server. And, to improve the performance, you want to run the batch script from a different server.

To run this batch file from a remote location, edit the batch file and update the paths to the Fast Data Masker .jar, connection file, masking file, and options file. Ensure that they all point to valid locations. You can then run the file from that remote location.

The following example snippet shows the contents of a masking batch file; update the required file locations based on your requirement:

```
java -Djava.util.logging.config.file="C:/Program
Files/Grid-Tools/FastDataMasker/logging.properties" -Xms1000M -Xmx10000M -jar
"C:/Program Files/Grid-Tools/FastDataMasker/Fastdatamasker.jar" "C:/Program
Files/Grid-Tools/FastDataMasker/doc/connectSQLSERVER.txt"
"C:\Users\<username>\AppData\Roaming\Grid-Tools\Fastdatamasker\MyMask.csv"
"C:\Users\<username>\AppData\Roaming\Grid-Tools\Fastdatamasker\MyMask_options.tx
t"
```

In this snippet, you can find the following example locations:

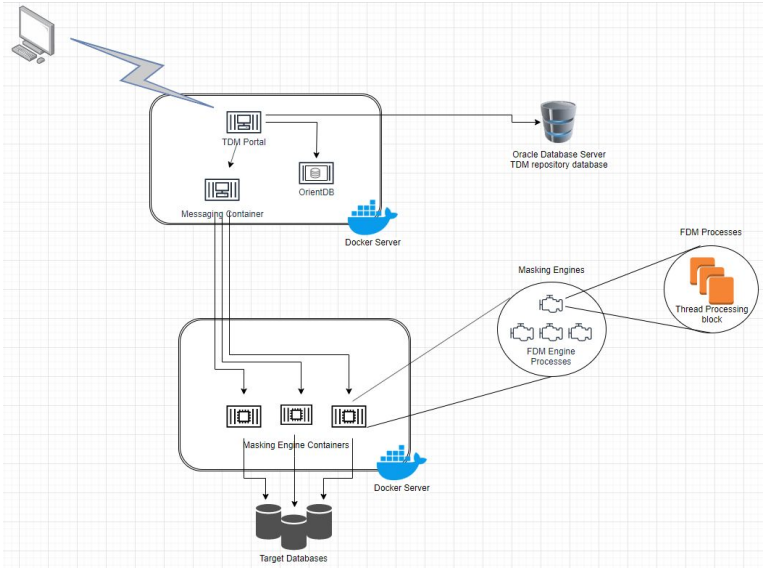
- C:/Program Files/Grid-Tools/FastDataMasker/Fastdatamasker.jar shows the location of the Fast Data Masker .jar file.
- C:/Program Files/Grid-Tools/FastDataMasker/doc/connectSQLSERVER.txt shows the location of the Microsoft SQL Server connection file.
- C:\Users\<username>\AppData\Roaming\Grid-Tools\Fastdatamasker\MyMask.csv shows the location of the file that contains masking information.
- C:\Users\<username>\AppData\Roaming\Grid-Tools\Fastdatamasker\MyMask_options.txt shows the location of the file that contains the applied options information.

Scalable Masking

Scalable masking deployment

The Scalable Masking feature was incorporated in CA TDM with version 4.8 and allows the scaling of masking jobs across multiple FDM masking engines. The masking engines are deployed in Docker containers and communicate with the TDM portal via a message bus which is also deployed in a docker container. The masking engines can be deployed on the same docker host as the messaging bus or remote docker hosts. When a new FDM engine is started it will automatically register with the messaging host. When using the scalable masking solution the TDM portal can be deployed on Windows or in a Docker container.

The TDM Docker masking containers can generate a high load when running multiple masking jobs so it is recommended to separate the docker masking engines from other TDM components. In the sample architecture below the TDM Portal / Messaging and OrientDB containers are deployed separately from the masking containers. The masking containers are configured to communicate with the messaging container on the primary TDM server.



TDM Masking Components explained

TDM Portal	TDM Manager and UI Server
TDM Repository	Oracle Database storing TDM data
OrientDB	TDM Internal Database (Models and Find and Reserve data)
Messaging Container	Communication bus between TDM Portal and Masking engines
Masking Engines Container	Docker container hosting FDM Engines
FDM Engines	Each Masking engine container can run four separate masking engines
FDM Processes	Each FDM Masking engine can spawn additional FDM Processes when using the large table masking functionality

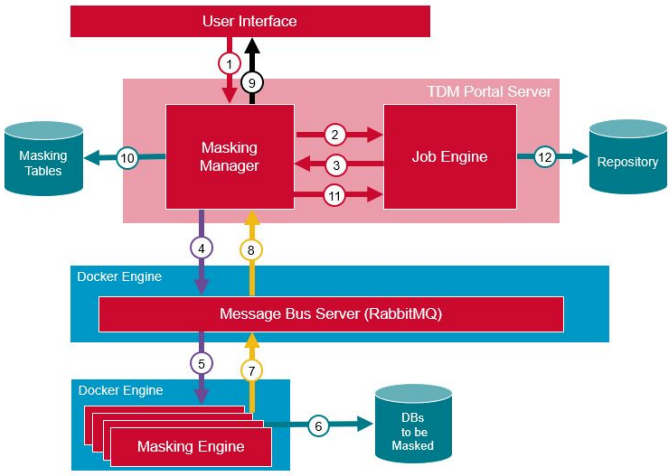
Scalable Masking Flow

1 User initiates masking job via REST request

2 Request is added to queue of job engine (allowing for scheduling)

3 At appropriate time, job engine passes request onto masking manager

4 Masking manager resolves job information, splits into tasks. one per schema, one task for each large table. These tasks are then added JMS queue



5 Masking engine pulls task from queue, each docker container contains four masking engines

6 Masking engine begins masking operations (using FDM engine). Threads can be used when setting Parallels

7 Masking engine provides ongoing status updates and final audit via JMS queue

8 Masking manager pulls status and audit messages from queue

9 Ongoing status is passed to UI via Websockets connection

10 After completion, final audit information is written to masking store

11 Job status is updated to the job engine via REST

12 Job status is written to the repository

Chapter 3 : Synthetic Data Generation Optimization

The development team performed testing to show performance benchmarks for publishing to CSV and XLSX files. You can use these benchmarks to help tune the performance of your system, Further details on the testing can be found on the Broadcom Docops site

Environment Details

The following environment was set up to gather performance data for publishing to CSV and XLSX files.

Machine specs

The machine has 16 GB physical memory, 4 vCPU, and runs Microsoft Windows Server 2012 R2 DataCenter. The repository was on a local SQL Server 2016.

Test 1 - Hard-coded Data

No expressions were used. All data was hard-coded in the generator.

Publish to CSV

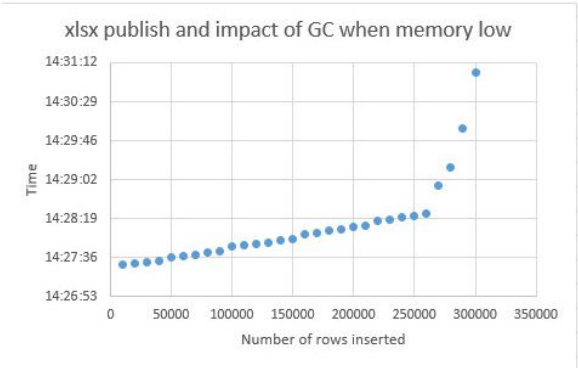
Repeater	Performance based on number of rows
10,000	0.5 seconds
100,000	1.2 seconds
1,000,000	7.6 seconds
10,000,000	1 min 08 seconds
100,000,000	11 min 48 seconds

Publish to XLSX

Publish to XLSX is memory intensive compared to publish to CSV. Using the default CA TDM Portal configuration, the publish hit a wall around 300,000 counts when the CPU usage went high and stayed high. In fact, performance started degrading around 260,000 counts.

This behaviour is caused by the GC (Garbage collector) going overdrive when trying to clean up some memory to make sure that the Portal application does not crash with an out-of-memory exception.

The following graph outlines the impact of garbage collection on the Portal:



Tip: Before starting a high-volume publish to XLSX, make sure you increase the memory that is allocated to the Portal.

Edit the config file called wrapper.conf located under CA\CA Test Data Manager Portal\service\conf. You can set either maxmemory or maxmemory.percent. With maxmemory.percent, the maximum allocated memory is calculated from the number that was set, times the physical memory. For more information about what needs to be done to increase the memory size used by the java process,
see https://wrapper.tanukisoftware.com/doc/english/prop-java-maxmemory.html.

Repeater	performance based on number of rows
100,000	32 seconds
200,000	59 seconds
300,000	1 min 30 seconds
400,000	1 min 49 seconds
500,000	2 min 40 seconds
600,000	3 min 36 seconds

Test 2 - One Expression

We publish using one expression ~NEXT~ in the generator.

Publish to CSV

Repeater	performance based on number of rows
1,000,000	14 seconds
10,000,000	1 min 29 seconds
100,000,000	12 min 38 seconds

Publish to XLSX

Repeater	performance based on number of rows
100,000	24 seconds
200,000	46 seconds
300,000	1 min 18 seconds
400,000	1 min 47 seconds
500,000	2 min 17 seconds
600,000	2 min 41 seconds
700,000	3 min 16 seconds

SQL Server Target publish (default config)

Repeater	performance based on number of rows
100,000	4 min 18 seconds
200,000	8 min 16 seconds
400,000	17 min 36 seconds
(portal restarted) 800,000	23 min 12 seconds

SQL Server Target publish (iterationsBeforeCommit=20000)

tdmweb.publish.batchCommit=true

tdmweb.publish.iterationsBeforeCommit=20000

Repeater	Performance based on number of rows
800,000	35 seconds
10,000,000	6 min 33 seconds
100,000,000	1 hour 6 min 46 seconds

SQL Server Target publish (iterationsBeforeCommit=50000)

tdmweb.publish.batchCommit=true

tdmweb.publish.iterationsBeforeCommit=**50000**

Repeater	Performance based on number of rows
1,000,000	42 seconds
10,000,000	6 min 31 seconds

Chapter 4: CA Test Data Manager Health Check

Health Check Overview

The health check should be used to gather data on the TDM environment and configuration:

1 Test Data Manager Challenges and pain points

_____

_____

_____

_____

2 Test Data Use Cases in use:

Data Masking

Data Subsetting

Data Cloning

Synthetic Data Generation

PII Discovery

Javelin Automation

Test Data Find and Reserve

vTDM

3 Test Data Architecture overview

_____

_____

_____

_____

CA Test Data Manager Health Checklist

1. Verify CPU Usage of TDM Server
2. Verify Memory Utilisation of TDM server
3. Verify Disk Space Usage of TDM Server
4. Review TDM Portal “startup.log” for errors
5. Run the TDM Repository maintenance utility
    a. For versions of TDM prior to TDM 4.8 the repository maintenance utility is launched from within DataMaker
    b. For TDM 4.8 installations using the utility schema-management.bat which can be found in the following default location :
        i. C:\Program Files\CA\CA Test Data Manager Portal\schema-management\bin
6. Review and document license keys
    a. TDM License
        i. For versions of TDM prior to TDM 4.8 the license can be reviewed in TDM Datamaker UI
        ii. For TDM 4.8 installations the license can be reviewed from the TDM portal
    b. FDM License
        i. License key is stored in this location C:\ProgramData\CA\TDM\lic.dat
    c. Subset

TDM Components Used

TDM Portal		
	Find and reserve	
	Masking	
	Data Modelling	
	TDod / ARD Forms	
DataMaker		
Fast Data Masker (FDM)		
GT Subset		
Javelin		

Data Sources in use

Document 3rd Party JDBC drivers for

Database	TDM Portal Find and Reserve	TDM Portal Data Modelling	TDM Portal Data Generation	Fast Data Masker	DataMaker Data Generation	Data Subsetting	Test Match	vTDM
Microsoft SQL Server								
Oracle								
DB2 for zOS								
DB2 iSeries								
Teradata								
Sybase								
PostgreSQL								
Adabas								
Informix								
Ingres								
MySQL								
MariaDB								

IMS								
Netezza								
SQL Anywhere								
Derby								
SQL Files								
CSV Files								
Fixed Definition files								
XML Files								
Excel Files								
Database	TDM Portal Find and Reserve	TDM Portal Data Modelling	TDM Portal Data Generation	Fast Data Masker	DataMaker Data Generation	Data Subsetting	Test Match	vTDM
TXT Files								
VSAM / ISAM								
JSON Files								

Environment Overview

	CPU	Memory	Disk Space
Environment			
TDM Portal			
DataMaker			
FDM			
Subset			

	MS SQL Version	Oracle Version	
TDM Repository			

List of Fast Data Masker (FDM) Functions used

_____

_____

_____

_____

_____

_____

TDM Usage data

Number of Published jobs Per Week	
Number of Published jobs Per Year	
Number of Find and Reserve Models	
Number of Subset Jobs per week	
Number of Subset jobs per year	
Number of Masking jobs per week	
Number of Masking jobs per year	

Chapter 5: Upgrade Process and Checklist

Upgrade Preparation and Backup	1	Create support ticket and request latest TDM build
	2	Download CA TDM installers from support.ca.com
	3	Backup latest TDM repository
	4	Backup TDoD Configurations using - Configuration Editor Backup option
	5	Backup Remote Publish Configurations using - Configuration Editor option
	6	Backup profiles.xml from %AppData%/Grid-Tools/ folder
	7	Backup rep.xml from %AppData%/Grid-Tools/ folder
	8	Delete rep.xml from %AppData%/Grid-Tools and C:\Program Files (x86)\Grid-Tools\GTDatamaker
	10	Backup C:\Program Files\CA\CA Test Data Manager Portal\conf\application.properties
	11	Backup OrientDB Databases C:\ProgramData\CA\CA Test Data Manager Portal\orientdb\databases
	12	Optional: If using custom SSL Certificates with TDM Portal, Backup certificates
TDM 4.8 Upgrade	13	Stop Remote Publish Service using - Configuration Editor
	14	Stop TDoD Service using - Configuration Editor
	15	Stop CA Portal Service
	16	Disconnect and close existing DataMaker windows in all the systems
	17	Start GT setup and install all the components as per documentation
	18	Run Portal installer to upgrade portal
	19	Start Portal Service
	20	Login to the portal server using the administrator account and activate the license
	21	Start Data Maker exe and connect to TDM repository
	22	Restart Data Maker --> connect to source/target profile and validate connections
	23	Start Remote Publish Service using - Configuration Editor
	24	Start TDoD Service using - Configuration Editor
	25	Connect to Data Maker and validate data pools publishes
	26	Connect to TDoD and validate tiles
	27	Stop Portal Service
	29	Start Portal Service