sitecore®

Sitecore 7.5

xDB™ Configuration Guide

Configuration guide for Sitecore administrators and developers

Table of Contents

Chapter 1

Introduction

The Sitecore® Experience Database™ (xDB™) collects all online and offline customer interactions from all channel sources in a real-time big data repository. It connects interaction data to create a comprehensive, unified view of each individual customer, and makes the data available to marketers to manage the customer experience in real time.

This document provides guidance on common configuration tasks for administrators or developers when setting up the xDB for the first time.

This guide contains the following chapters:

- Introduction

- Standard Configuration

- Cloud Configuration Options

- Customization

1.1 Architecture Options

When you install the Sitecore® Experience Platform™ (Sitecore® XP) with the xDB for the first time there are several different architecture options that you can choose from. The most common scenarios are outlined below:

1.1.1 On-Premise Server Solutions

All xDB components, such as Sitecore application servers and MongoDB are installed on the customer's own servers. There are two ways in which you can install an on-premise Sitecore solution:

- **Single server standalone environment (not scalable)**. All components installed on a single server computer.

- **Multi-server scalable environment (fully scalable)**. A multi-server environment consists of dedicated servers for each Sitecore component. For example, content delivery, content management, collection database, session state, reporting service, reporting database and processing servers. You also have the option of expanding this to handle big data by implementing vertical and horizontal scaling.

Note
Only install a standalone environment for development or testing purposes. For a production environment we do not recommend that you install all xDB components together on a single server.

Advantages and disadvantages of on-premise Sitecore server solutions:

Advantages	Disadvantages
More control	Complicated hardware setup.
Unlimited configuration options.	Costly to create and maintain.
Potentially unlimited hard disk space depending on budget.	Time consuming to administer.
Easier to choose whether to use SSD or HDD drives.	Costly and time consuming to expand/upgrade and scale storage.
	Requires multiple data centers for geographically distributed failure zones. This is much easier and cheaper to achieve with the public cloud.

Whether you choose to run Sitecore in the cloud, as an on-premise solution or a hybrid solution depends on the type and size of organization. For example, a small organization may only need a few servers and therefore can use a public cloud solution whereas a larger enterprise organization may have its own data center.

Note
Public cloud providers can also offer unlimited disk space. The limitation in both cases is the budget. It is much easier and faster to scale storage in the public cloud than within your own premises.

1.1.2 Hybrid Server Solutions

In a hybrid solution some components are installed as part of an on-premise solution while others are installed in the cloud.

For example:

- Multi-server scalable Sitecore environment installed as an on-premise solution (see the previous section for more details). This can include content delivery, content management, processing, reporting service and reporting database servers installed as an on-premise solution at an organization's own data center.

- MongoDB server installed in the cloud using Sitecore xDB Cloud Edition (see the next section for details).

Advantages and disadvantages of hybrid solutions:

Advantages	Disadvantages
Hosting MongoDB in the cloud removes the need for specialist MongoDB knowledge and expertise. Therefore no new staff or training is required.	You have less control over scaling the collection database to handle large numbers of contacts and visits.
Ease of maintenance and upgrade.	There may be limitations when using MongoDB with some private cloud solutions. For example, a limited amount of data storage.

1.1.3 Cloud Server Solutions

In a full-scale cloud solution all Sitecore components are installed in the cloud.

For example:

- Fully scalable cloud environment – this includes all Sitecore xDB components, such as content delivery, content management, processing, reporting service, reporting database, session state and MongoDB (collection database).

Advantages and disadvantages of cloud solutions:

Advantages	Disadvantages
Simplifies installation and configuration.	Cost.
Less specialist knowledge is needed.	Less control - You have less control over the hardware you use and how you handle scalability.
Data centers can be distributed in remote locations around the globe ensuring high availability.	
	It may not suit very large organizations which have their own data centers to have all their servers hosted in the cloud. Then a hybrid solution may be a good option.
Backup is done for you.	
You don't need to manage scalability.	
Security. You can spread data across multiple locations.	

For more information on architecture see the *xDB Overview and Architecture document.*

For more information on Cloud options, see *xDB Cloud Quick Start Guide* on SDN or contact your local Sitecore office.

Chapter 2

Standard Configuration

This chapter covers all the basic steps and guidance you need when setting up the Sitecore Experience Platform as an on-premise solution for the first time.

This chapter contains the following sections:

- Hardware Guidelines

- Software Recommendations

- Installation

- Configuring Connection Strings

- Server Configuration

- Session State

2.1 Hardware Guidelines

When you install Sitecore xDB™ as an entirely on-premise solution you need to consider the minimum hardware requirements for each instance in your environment depending on the roles you have allocated to each server.

Content Management and Content Delivery Servers

The requirements for the content management and content delivery servers have not changed.

For more information on specific requirements, see the *Sitecore CMS 7.5 Installation Guide* on SDN.

Session State Server

The hardware requirements for the session state server depend on which session state service you choose to use. If you choose to install a session state server with the xDB, you can configure this to use a MongoDB database.

We recommend you use a fast network, solid-state drive (SSD) on your session state server and plenty of RAM.

For more information on these requirements, see the *MongoDB installation guide* on the MongoDB website: MongoDB installation guides

Collection Database Server

Using MongoDB as your collection database, we recommend that you install plenty of RAM and use SSD drives. Sharding can also improve performance significantly.

For more information on MongoDB architecture, replication and configuration options, see the MongoDB documentation on the MongoDB website, the *xDB Configuration Guide* and the *xDB Overview and Architecture* document.

Processing and Aggregation Server

Processing and aggregation servers can use RAM for caching and for establishing a fast connection to the databases. The more cores the CPU has, the more agents you can configure to run in parallel.

Reporting Server

The reporting server has the same requirements that the analytics database server had in previous versions of Sitecore. There are no significant changes compared to Sitecore CMS and DMS 7.1.

Better optimization of the reporting database means it now uses memory more efficiently. However, install fast hard drives and plenty of RAM to achieve the best performance.

For more detailed information, see *Microsoft SQL Server documentation* on MSDN*.*

2.1.1 Hardware Configuration Example from Sitecore.net

The following hardware example setup described in this section was implemented on the Sitecore.net website when setting up the xDB for the first time. There are many alternative ways of installing and configuring Sitecore xDB and this is just one example.

Important Note
The hardware details outlined in this section are an example. They are **not** a set of recommendations to follow.

You can use the information in this section to get an idea of what you might need to consider when setting up the xDB for the first time.

Example Hardware Configuration (Sitecore.net):

Server type	CPU	RAM	Disk
Content delivery	4 x CPU E5 2650 v2 processors with the more cores the better.	16 GB (including the operating system).	HDD 40 GB per instance.
Processing	4 x CPU E5 2650 v2 processors with the more cores the better.	16 GB (including the operating system).	HDD 40 GB per instance.
Content management	4 x CPU E5 2650 v2 processors with the more cores the better.	16 GB (including the operating system).	HDD 40 GB per instance.
Session state (MongoDB)	4 x CPU E5 2650 v2 processors.	4 GB The more the better.	SSD shares hard disk with the collection database, ideally the hard disk capacity should not be more than the RAM which is 4GB in this example.
2 x collection (MongoDB) servers	4 x CPU E5 2650 v2 processors.	16 GB The more the better.	100 GB + SSD The more disk space you have, the more data you can store.
Reporting database server (SQL Server)	4 x CPU E5 2650 v2 processors.	16 GB The more the better.	100 GB + SSD The more the better.

HDD = hard disk drive

SSD = solid state drive

RAM

Operating System

Include the Windows operating system (OS) in the RAM usage because it is difficult to predict accurately how much RAM the Windows OS will consume.

Session

The more RAM you have, the more sessions you can track simultaneously without using extra disk space. Actual performance though depends on the specific configuration you have. Disk I/O on reads should be avoided.

Session database

The session database should ideally fit 100% into RAM, otherwise it gets too slow.

Collection database (MongoDB)

RAM serves as disk cache. You need at least enough RAM to store working set size (MongoDB has instructions on how to estimate these, Microsoft SQL Server also provides this information).

Hard Disk

In general, the more disk space you have, the more data you can store.

Note
These example recommendations aim for achieving the best performance possible but have not been tested for every possible scenario. Where we say, "*the more the better*", this means that you need to perform tests to find out what suits your system best.

2.2 Software Recommendations

For each xDB server instance that you install you need the following minimum software prerequisites:

- Microsoft .NET Framework 4.5.

- Windows Server 2012 R2 – recommended for most Sitecore instances, especially processing/aggregation servers.

- MongoDB 2.6.1 or later for the collection database, session storage and tracking databases. Install MongoDB on Microsoft Windows Server as a Windows service.

- Microsoft SQL Server 2008 R2 SP1 or higher for the reporting database.

For more detailed information on the specific software requirements for standard Sitecore CMS instances, see the *Sitecore CMS 7.5 Installation Guide* on the Sitecore Developer Network (SDN).

2.3   Installation

This section provides general guidance for administrators on installing the Sitecore Experience Database (xDB) with MongoDB for the first time.

For a complete set of steps and more information, see the *Sitecore 7.5 Installation Guide* on SDN.

2.3.1   Install Sitecore CMS

You can install Sitecore CMS using the Sitecore installation program or install it manually from a .ZIP archive. The installation program automatically installs a Sitecore client and databases. It does not install the MongoDB collection database as part of the installation.

Note

When you open Sitecore for the first time you may get an error if you have not updated the `ConnectionStrings.config` file to point to the collection database (MongoDB). In the connection strings configuration file the collection database has the name `analytics`.

For more information on configuring connection strings, see the section Configuring Connection Strings.

For complete instructions on installing Sitecore CMS, see the *Sitecore 7.5 Installation Guide* on SDN.

2.3.2   Install Microsoft SQL Server

Install Microsoft SQL Server 2008 R2 SP1 or higher and SQL Server Management Studio.

For more detailed information on installing SQL Server, see the *Sitecore 7.5 Installation Guide* on SDN and Microsoft SQL Server installation instructions on MSDN.

2.3.3 Install MongoDB

Install MongoDB as your NoSQL database server.

MongoDB Considerations

Before you start installing a MongoDB database you should consider the following:

- Decide whether you want a public cloud-based solution such as Windows Azure or one within your own data center on either physical or virtual hosts.

  http://docs.mongodb.org/manual/faq/fundamentals/

- Ensure that the MongoDB working set fits in memory for optimal performance. This is usually the most important performance scale up factor, followed by disk and CPU.

  http://docs.mongodb.org/manual/faq/storage/#what-is-the-working-set

  http://docs.mongodb.org/manual/faq/diagnostics/#how-do-i-calculate-how-much-ram-i-need-for-my-application

- Selecting hard disks – use solid-state drives (SSD) for fast reads and writes or hard disk drives (HDD) for larger capacities at lower cost. SSDs are recommended as they offer significantly higher performance. If HDDs are used, we recommend 15k RPM SAS disks in a RAID10 configuration for a balance of performance and fault tolerance.

- Decide how many MongoDB servers you need:

  o   Standalone server – for testing and development.

  o   Single replica set – The minimum recommended configuration for production, which should have at least 2.5 servers: two full capacity data servers for failover and one low

capacity server for the arbiter. We recommend three data servers for a robust and resilient deployment, especially during maintenance.

http://docs.mongodb.org/manual/replication/

Sharded cluster – When and how to implement this depends on the data size and performance requirements. There will be multiple shards, with each being a replica set.

http://docs.mongodb.org/manual/core/sharding/

For more information, see the *Performance Considerations for MongoDB* and *MongoDB Operations Best Practices* whitepapers on the MongoDB website.

http://info.mongodb.com/rs/mongodb/images/MongoDB-Performance-Considerations_2.4.pdf

http://info.mongodb.com/rs/mongodb/images/10gen-MongoDB_Operations_Best_Practices.pdf

Note
We recommend that you only use single server installations for development and testing.

MongoDB Installation

Install an empty MongoDB database following the installation instructions provided on the MongoDB website: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

When you have completed the installation of Sitecore and MongoDB you will not see any collections or documents in MongoDB straight away. MongoDB only creates the required database collections and documents you need when there have been visits to your website.

Note
In MongoDB you carry out most administration tasks using the command line tool mongo shell. If you prefer to use a GUI administrative interface, you could use a GUI client such as MongoVUE but note that this tool is not written or supported by MongoDB, Inc.

Configuring MongoDB as a Service

When you install MongoDB, configure it as a Windows service so that the Mongo service starts automatically every time you start your computer. If you wish, you can change this setting in *Services* to start the Mongo service manually. See the MongoDB documentation for steps on how to do this using the command line:

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

Connecting to MongoDB

You can only connect to MongoDB if you have first configured the connection strings correctly in the `ConnectionStrings.config` file in your site root. For more information on configuring connection strings see the section Sitecore Experience Platform Connection Strings.

After you have configured your connection strings, test MongoDB by opening the MongoDB client and connect to the MongoDB database that you created.

To connect to MongoDB using MongoVUE:

1. In the MongoVUE client, click *Connect* to create a connection to your MongoDB installation.
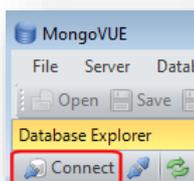
2. In the **Connect** window, click the **+** symbol to open the **Create new Connection** dialog box.

3. Enter a name for your MongoDB instance, a server name, and a port number.

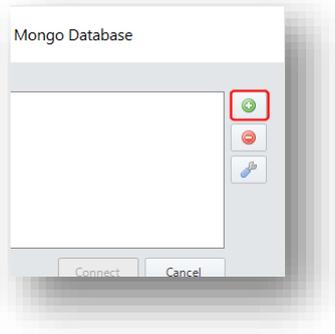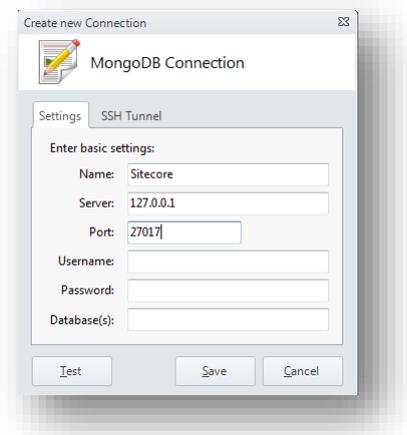4. Click *Test* to test the connection and click *Save* to save your connection settings.

   Once you have created a connection, you should be able to see the databases with the names that you specified in the `ConnectionStrings.config` file.

   MongoDB databases:

   - `analytics`
   - `anayticshistory`
   - `analyticslive`
   - `local`

   Note
   You need to start Sitecore and visit your website before these databases appear in MongoDB.

For an example MongoDB replica set architecture, see the *xDB Overview and Architecture* document.

2.4 Configuring Connection Strings

This section contains information about the connection strings you need to configure in Sitecore® XP. For more detailed information on configuring connection strings for dedicated servers, see the section Server Configuration.

For more information on configuring connection strings when installing Sitecore® XP, see the *Sitecore CMS 7.5 Installation Guide* on SDN*.*

Sitecore Experience Platform Connection Strings

After you have installed Sitecore CMS, configure the CMS connection strings and any other connection strings you need.

How you configure connection strings can vary depending on the role/s you have assigned to your servers, for example, whether you have dedicated servers for content management, content delivery or for processing. This section describes the general settings you need for a minimal environment and not the specific parameters you need to configure each role.

To configure Sitecore connection strings open the `ConnectionStrings.config` file:

1. In your website root folder navigate to the **App_Config** folder:

   `<sitename>\Website\App_Config`

2. Open the `ConnectionStrings.config` file in Visual Studio.

3. Configure CMS connection strings for the *Core*, *Master,* and *Web* databases, MongoDB for the collection database and SQL Server for reporting.

4. See the table below for example connection strings and instructions on how to do this.

5. Save all changes to the `ConnectionStrings.config` file.

Database Name	Description

Core
Master
Web
	Configure the Sitecore CMS, SQL Server connection strings for the *Core*, *Master,* and *Web* databases.
	See the example connection strings below:
	
	`<add name="core" connectionString="user id= sql server user ;password= user password ;Data Source=_sqlserver_;Database=_core_database_name_" />`
	
	`<add name="master" connectionString="user id= sql server user ;password= user password ;Data Source=_sqlserver_;Database=_master_database_name_" />`
	
	`<add name="web" connectionString="user id=_sql_server_user_;password=_user_password_;Data Source=_sqlserver_;Database=_web_database_name_" />`

Database Name	Description
Collection	This is a NoSQL MongoDB database. Mongo DB collection databases include: • `analytics` • `tracking.live` • `tracking.history` Configure these connection strings to connect to MongoDB: ``` <add name="analytics" connectionString="mongodb://_mongo_server_name_:_port_number_/_ collection_db_name_" /> <add name="tracking.live" connectionString="mongodb:// mongo server name : port / tracking live db name " /> <add name="tracking.history" connectionString="mongodb:// mongo server name : port / tracking.history db name " /> ```
Session	If you choose to use a session state server, then configure one of the following connection strings (this is an optional step). MongoDB: ``` <add name="session" connectionString="mongodb:// mongo server name : port /_session_db_name_" /> ``` SQL Server: ``` <add name="session-mssql" connectionString="user id=_sql_server_user_;password=_user_password_;Data Source=_sqlserver_;Database=_session_db_name_" /> ``` .
Reporting	This is a SQL Server database. The reporting database contains *Fact* and *Dimension* tables for storing all aggregated data used by Sitecore reporting applications. ``` <add name="reporting" connectionString="user id= sql server user ;password= user password ;Data Source=_sqlserver_;Database=Sitecore_Reporting" /> ```

Database Name	Description
Reporting secondary	This is a SQL Server database. Use the reporting secondary database if you want to perform a rebuild of the reporting database. Reasons for performing a rebuild of the reporting database: • If you have re-classified a search key word or traffic type, aggregated report data is not updated automatically. • To keep the collection (MongoDB) and reporting databases (SQL) synchronized with each other. `<add name="reporting.secondary" connectionString="user id=_sql_server_user_;password=_user_password_;Data Source=_sqlserver_;Database=Sitecore_Reporting_Secondary" />` **Note**: You only need to connect the reporting secondary database if you want to perform rebuild of reporting database.

6. Once you have configured all your connection strings and added the configuration files, test your Sitecore installation by launching your website and you should see a blank Sitecore home page.

7. To verify that you have connected to the reporting database open SQL Server Management Studio and expand the **Tables** node to see all the Fact and Dimension tables contained in the reporting database.
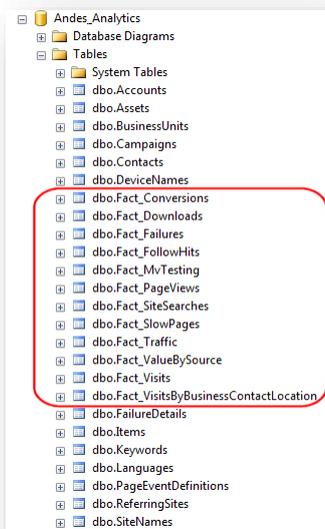
8. Test the MongoDB installation by checking that all your xDB databases, such as *analytics* and *tracking_live* appear in your MongoDB client.

After configuring connection strings for MongoDB you may need to generate some data on your website before you see any collections or documents. MongoDB creates the databases dynamically, so if your website has no contact or interaction data then you do not see any collections or documents.

For more information on testing your configuration, see Verifying a Connection to the Collection Database.

2.4.1   Verifying a Connection to the Collection Database

Follow the steps in this section to verify that you have configured your standalone Sitecore environment correctly.

To test that you have configured your Sitecore CMS connection strings correctly open your website in a web browser and check that you can log in to the Sitecore Desktop.

When you install the xDB for the first time there is no pre-defined website. To test your connection to MongoDB, generate some local visits to your empty website. Any visits you make to your website appear as new interactions in the collection database.

Verify the MongoDB Collection Database

To check that the interaction appears correctly in the collection database:

1. Open your NoSQL database, open the MongoDB client and connect to the appropriate database.

2. In MongoVUE, you can see that before any contacts visit your website, only some collections are visible. In the following screenshot, the Analytics database appears but it does not yet contain any collections.
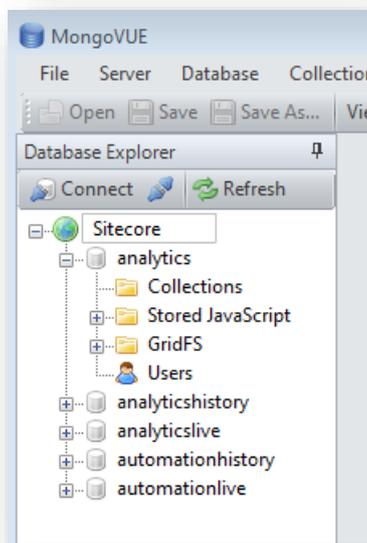
The *Contacts* and *Interactions* collections are missing until your website has visits.

Make a visit to your website test page then refresh MongoDB. Expand the *analytics* database and then expand the **Collections** node.
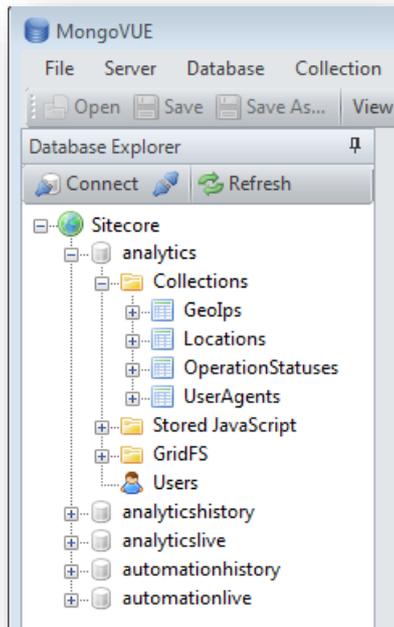
You can see four new collections relating to browser and system information but no interaction or contact collections appear yet.

Session End Default Timeout Setting

After making a test visit, you may need to wait 20 minutes before MongoDB updates with the latest changes. For testing purposes you may wish to make data appear more quickly.

To change the default timeout setting:

1. Open the `web.config` file.

2. Navigate to the `<sessionState` node.

3. Change the session end default timeout setting from 20 to 1 minute.

```
<sessionState mode="InProc" cookieless="false" timeout="1">.
```

4. Make a visit to your website test page then refresh MongoDB. Data should now appear in MongoDB after one minute.

   Note
   It is highly recommended that you change the default timeout setting for testing purposes only. The default setting should be restored as soon as possible.

5. Refresh the MongoDB database again. After a short time you should see the *Contacts*, *Devices*, and *Interactions* collections added to MongoDB.
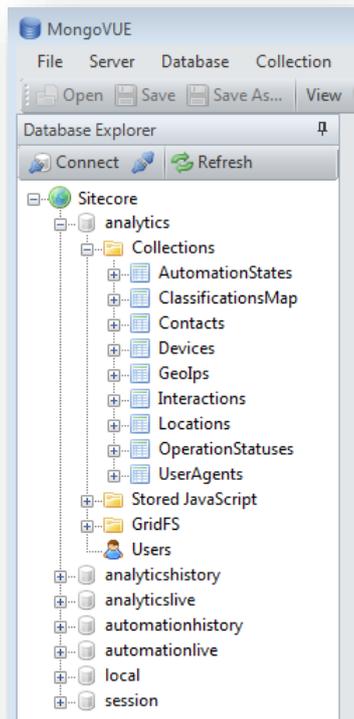
6. Right click *Interactions* and then click *View* to see all the interactions made with your website.

7. In this example, five interactions are stored in this collection. Expand one of these interactions and view data relating to the pages visited. The visit in the following screenshot consists of

two page views.
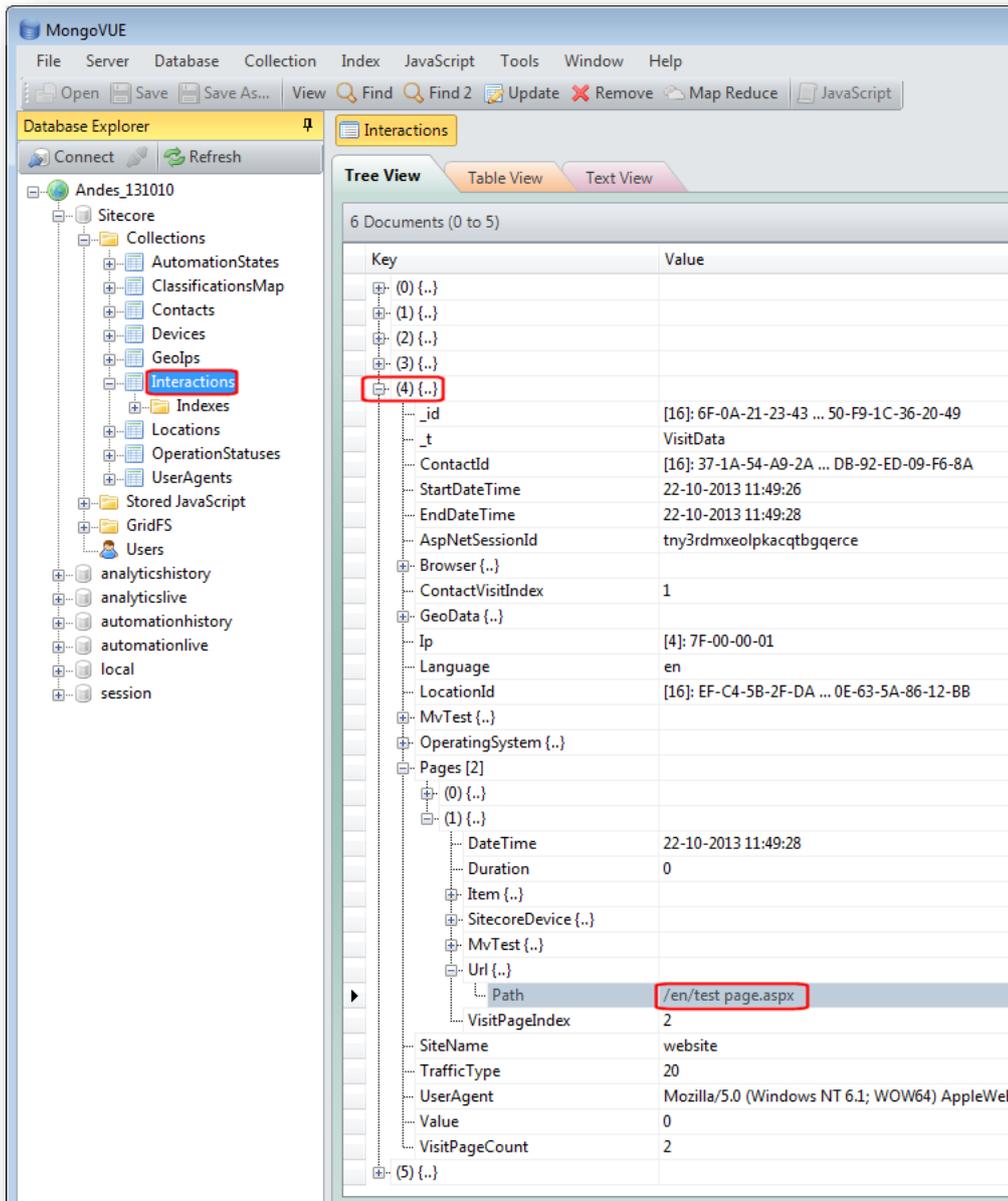
Verifying SQL Server Fact and Dimension tables

Before you can verify that *Fact* and *Dimension* tables contain pre-aggregated data, ensure that you have data in the collection database and have configured processing correctly.

For more information on configuring a standalone processing server, see section 3.3.4 **Error! Reference source not found.**.

Query one of the SQL Server *Fact* tables, such as `Fact_PageViews` and view all rows to see if it contains any data. If the table contains analytics data then the processing layer is working correctly.
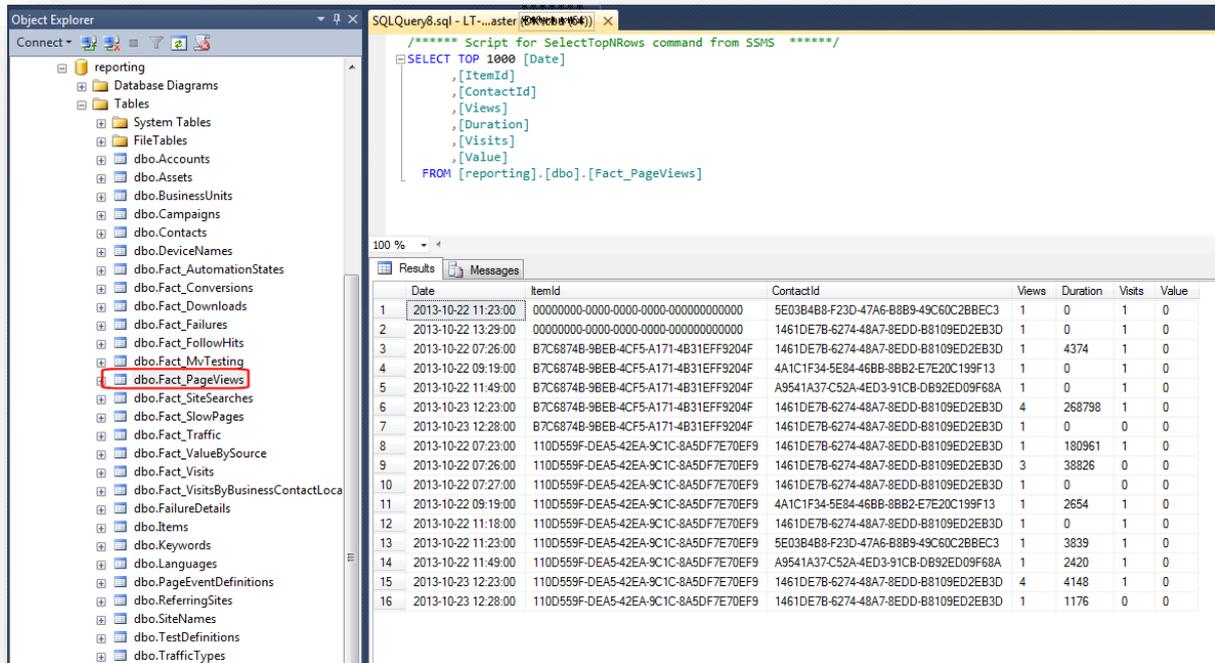
Note

If you cannot see any data in the *Fact_* tables check that the *ConnectionStrings.config* on the processing server to see if the *analytics* connection string to MongoDB is configured correctly and that you have data in the collection database.

2.5 Server Configuration

Just like its predecessor, the xDB supports a standalone all-in-one configuration, retaining the ability to run on a single server. This allows for easy deployment of a development environment or a small, low-traffic production instance. However, you only see the real benefits and flexibility of the xDB architecture when you start trying to improve the performance of your application.

To achieve better performance and scalability you can configure dedicated servers for different purposes such as for content delivery or data processing. You can choose from a list of server features that you can configure on single or multiple dedicated servers.

In practical terms, to assign features to a dedicated server you simply configure a reduced set of components on a server. You can do this by disabling or enabling configuration files in your Sitecore installation. This gives you flexibility and fine-grained control over the tasks and activities that you assign to each of your Sitecore instances.

By dividing your Sitecore application up into multiple servers performing different activities, you increase your scaling options. For improved application performance, you can scale both vertically (by scaling up the hardware on each server) and horizontally (by adding more servers to perform each activity). For improved availability and reliability, you can create clusters, such as for content delivery or data processing.

For additional flexibility, you can also configure a single server to perform multiple server tasks. For example, you could enable content management, content delivery, and reporting on a single server but move the MongoDB collection database to another server.

2.5.1 Server Features

To configure a server to use the following server features, you may need to make changes to the default configuration of your Sitecore solution.

The following table describes the server features that you can enable or disable on dedicated servers along with the required connection strings and configuration files that you need to enable or disable. All connection strings are stored in the `ConnectionStrings.config` file which is located in the `/App_Config/` folder and all configuration files are stored in the `/App_Config/Include` folder.

Server Feature and Description	Connection strings	Configuration files
Tracking To track online visits, contacts (visitors), personalization, goals, campaigns, profiles, pattern cards and multivariate tests. All tracking data is saved by default in the MongoDB collection database.	Configure the following database connection strings: • analytics • tracking.live	Required configuration files: • `Sitecore.Analytics.Tracking.config` • `Sitecore.Analytics.Tracking.Database.config` • `Sitecore.Analytics.Tracking.Aggregation.config` • `Sitecore.Analytics.Tracking.RobotDetection.config` If you want to enable all tracking subsystem features but don't want tracking data to be saved to the collection database, you must disable the following configuration files: • `Sitecore.Analytics.Tracking.Database.config` • `Sitecore.Analytics.Tracking.Aggregation.config` • `Sitecore.Analytics.Tracking.RobotDetection.config`
Reporting Can use data from the collection and reporting databases. Data is used in applications such as Engagement Analytics reports and the Executive Insight Dashboard.	Configure the following database connection strings: • analytics • reporting	Required configuration file: • `Sitecore.Analytics.Reporting.config`
Processing Processing can rebuild the entire reporting database on request if data in the collection database has changed and has become out of sync. For example, classification of contact or locations.	Configure the following database connection strings: • tracking.live • tracking.history • analytics • reporting.secondary	Required configuration file: • `Sitecore.Analytics.Processing.config` • `Sitecore.Analytics.Processing.Services.config`

Server Feature and Description	Connection strings	Configuration files
Aggregation Responsible for extracting data from the collection database, grouping it and reducing it and then storing it in the reporting database for use by Sitecore reporting applications. Aggregation keeps the reporting database synchronized with the collection database.	Configure the following database connection strings: • tracking.live • reporting	Required configuration files: • `Sitecore.Analytics.Processing.Aggregation.config` • `Sitecore.Analytics.Processing.Aggregation.ProcessingPools.config` • `Sitecore.Analytics.Processing.Aggregation.Services.config`
Reporting Service Can query different data sources for Sitecore reporting applications: it queries the reporting database for grouped data, aggregated data and trends and the collection database for individual contact or interaction data.	Configure the following database connection strings: • reporting • analytics	Required configuration files: • `Sitecore.Analytics.Reporting.config` • `Sitecore.Analytics.Reporting.RemoteClient.config` • `Sitecore.Analytics.Reporting.RemoteServer.config`
EAS Processing Processing of engagement automation state timeouts This feature is responsible for setting up the automation workers to process EAS timeout conditions.	Configure the following database connection strings: • analytics	Required configuration file: • `Sitecore.Analytics.Automation.TimeoutProcessing.config`

The following configuration files are necessary for all the server features and should therefore not be disabled:

- `Sitecore.Analytics.config`

- `Sitecore.Analytics.Model.config`

- `Sitecore.Analytics.Mongodb.config`

Note
All configuration files are enabled by default with the exception of
`Sitecore.Analytics.Reporting.RemoteClient.config` and
`Sitecore.Analytics.Reporting.RemoteServer.config` file, which are disabled by default.

2.5.2 Server Configuration Examples

This section includes step by step instructions on setting up dedicated servers to implement different server features. These steps may require you to make change to the default Sitecore xDB installation.

The examples in this section are standard server examples provided by Sitecore. This type of server configuration is most suitable for installations where you want to scale vertically. You can use these as examples as a guide or you can modify them by combining servers and features in a way that best suits your own Sitecore solution.

Note
When you install Sitecore, some configuration files are enabled by default. You may need to manually disable configuration files depending on the purpose of your server.

Content Delivery Server

A content delivery server requires the *Tracking* server feature.

To configure a single server for content delivery:

1. Install and configure your content delivery Sitecore environment according to the instructions in the Sitecore Installation Guide.

2. Remove or restrict access to the client. You do not need the Sitecore client on a content delivery server. For more information on how to do this, see chapter 4 in the Sitecore CMS 7.0 Scaling Guide on SDN.

3. Navigate to the `/App_Config/Include` folder and disable the following configuration files. To disable a configuration file change the extension from `.config` to `.disabled`.

   o `Sitecore.Analytics.Automation.TimeoutProcessing.config`

   o `Sitecore.Analytics.Processing.Aggregation.Services.config`

   o `Sitecore.Analytics.Processing.Services.config`

   o `Sitecore.Analytics.Reporting.config`

   Note
   When you install Sitecore, some configuration files used for content delivery are enabled by default.

4. Ensure that the following configuration files remain disabled:

   o `Sitecore.Analytics.Reporting.RemoteClient.config.disabled`

   o `Sitecore.Analytics.Reporting.RemoteServer.config.disabled`

5. Content delivery servers do not require all connection strings to be active. For example, there are some security benefits in removing the reporting connection string. Either remove these strings completely or add comment tags to hide them.

   In the `/App_Config/ConnectionStrings.config` file remove the following connection strings:

   ```
   <add name="reporting" connectionString="….."/>

   <add name="tracking.history" connectionString="mongodb:….."/>
   ```

6. In the `/App_Config/ConnectionStrings.config` file, configure the Sitecore CMS web application to connect to the Core and Web databases from the database server, but not the Master database.

   Note
   To disable the master database, you comment out the connection string.

7. Assign a name for each content delivery server. This is especially important if you are creating a cluster of content delivery servers. Add the name to the `Sitecore.Analytics.Tracking.config` file.

To assign a name, open the `Sitecore.Analytics.Tracking.config` file and add a name to the value parameter in the following setting:

```
<setting name="Analytics.ClusterName" value="" />
```

The name should be identical for all content delivery servers that are in the same cluster.

For example:

```
<setting name="Analytics.HostName" value="www.domain.com" />
```

```
<setting name="Analytics.ClusterName" value="cluster1.domain.com" />
```

Additional steps necessary to configure `Sitecore.Search` functionality:

1. Ensure that the `Sitecore.ContentSearch.Lucene.Index.Master.config` file is disabled.

2. Locate the `Sitecore.ContentSearch.DefaultConfigurations.config` file and comment out all strategies where the master database is used.

3. Locate the `Sitecore.ItemBuckets.config` and comment out the following section: `<database id="master">`.

4. At `Sitecore.ContentSearch.config` disable the following section by commenting it out:

```
<scheduling>
  <!-- An agent to optimize the specified indexes periodically. -->
  <agent type="Sitecore.ContentSearch.Tasks.Optimize" method="Run"
    interval="01:00:00">
    <indexes hint="list">
        <index>sitecore master index</index>
    </indexes>
  </agent>
</scheduling>
```

In this type of configuration, the content delivery server is used for tracking. We recommend that tracking is the only analytics feature you enable on a dedicated content delivery server.

Note
If you only have a single cluster then the cluster configuration described in this section is not applicable.

For more information on enabling other features on a server, see Server Features.

Note
To use Sitecore MVC with the xDB, navigate to `/App_Config/Include/Sitecore.MvcAnalytics.config.disabled`. Rename the `Sitecore.MvcAnalytics.config.disabled` file to `Sitecore.MvcAnalytics.config`.

Note
If you experience contact locking while running the Email Experience Manager (EXM) in a scaled environment, then see the ECM Administrator's and Developer's Guide for further tips and guidance.

Content Management Server

A content management server requires the *Reporting* server feature.

To configure a single server for content management:

1. Install and configure your content management Sitecore environment according to the instructions in the Sitecore Installation Guide and the Sitecore Scaling Guide.

2. Navigate to the `/App_Config/Include` folder and disable the following configuration files. To disable a configuration file change the extension from `.config` to `.disabled`:

- o `Sitecore.Analytics.Automation.TimeoutProcessing.config`

- o `Sitecore.Analytics.Processing.Aggregation.Services.config`

- o `Sitecore.Analytics.Processing.Services.config`

- o `Sitecore.Analytics.Tracking.Database.config`

- o `Sitecore.Analytics.Tracking.Aggregation.config`

3. Ensure that the following configuration file remains disabled:

- o `Sitecore.Analytics.Reporting.RemoteClient.config.disabled`

- o `Sitecore.Analytics.Reporting.RemoteServer.config.disabled`

4. Open the `/App_Config/ConnectionStrings.config` file and configure connection strings for the following databases:

SQL Server:

- o core

- o master

- o web

- o reporting

MongoDB:

- o analytics

- o tracking.live

Note

You should always configure the *sessions* connection string unless you use the InProc mode for session state. It is ok to configure a content management server with the InProc session provider because it does not perform the same tasks as a content delivery server. This also means a content management server should never be a part of a content delivery cluster.

In this type of configuration, the content management server is used for tracking but does not save any changes to the collection database.

Note
Before running history processing using the RebuildReportingDb.aspx page, make sure you add a reporting.secondary connection string to the `ConnectionStrings.config` file.

Example reporting.secondary connection string:

`<add name="reporting.secondary" connectionString="…."/>`

Note
If you are configuring a pure content management server role, the `Tracking.Database.config` file must be disabled. However, if you want the content management server role to be mixed, or if there is a chance that other modules may depend on the tracking data access API, then the `Tracking.Database.config` file must be enabled.

For more information on enabling other features on a server, see Server Features.

Note
To use Sitecore MVC with the xDB, navigate to `/App_Config/Include/Sitecore.MvcAnalytics.config.disabled`. Rename the `Sitecore.MvcAnalytics.config.disabled` file to `Sitecore.MvcAnalytics.config`.

Processing Server

Configure a processing server in the same way as you configure a content delivery server. You can create one or multiple processing servers that can be configured for aggregation or other kinds of processing. As a general rule, the more aggregation server instances you create, the faster aggregation processing will be.

A processing server can use one or more of the following server features:

- *Processing*
- *Aggregation*
- *EAS Processing*

You can configure separate servers to perform each of these tasks independently on dedicated servers.

To configure a processing server role:

1. Install a standard Sitecore CMS instance on the server you want to use for processing.

2. Remove or restrict access to the client. You do not need the Sitecore client on a processing server.

   For more information and steps on how to do this, see the *Sitecore CMS 7.0 Scaling Guide* on SDN.

3. Navigate to the `/App_Config/Include` folder and disable the following configuration files. To disable a configuration file change the extension from `.config` to `.disabled`:

   - `Sitecore.Analytics.Tracking.Database.config`
   - `Sitecore.Analytics.Tracking.config`
   - `Sitecore.Analytics.Reporting.config`
   - `Sitecore.Analytics.Tracking.RobotDetection.config`
   - `Sitecore.ContentSearch.Lucene.Index.Web.config`

   Ensure that the following configuration files remain disabled:

   - `Sitecore.Analytics.Reporting.RemoteClient.config.disabled`
   - `Sitecore.Analytics.Reporting.RemoteServer.config.disabled`

4. In the `/App_Config/ConnectionStrings.config` file comment out or remove the Web and Session connection strings:

   ```
   <add name="web" connectionString="user
   id= sql server user ;password= user password ;Data
   Source=_sqlserver_;Database=Sitecore_web" />
   ```

   ```
   <add name="session"
   connectionString="mongodb://_mongo_server_name:_port_number/_session_database_name" />
   ```

   In the `Sitecore.Buckets.config` file, comment out the following section:

   ```
   <database id="web" singleInstance="true" type="Sitecore.Data.Database,
   Sitecore.Kernel">
    </database>
   ```

5. Locate the `Sitecore.ContentSearch.DefaultConfigurations.config` file and comment out all the strategies where the Web database is used.

6. In the `web.config` file, comment out the `Sitecore.Tasks.PublishAgent` section.

7. In the `Sitecore.WebDAV.config` file, comment out the following section:
   `Sitecore.Tasks.CleanupFDAObsoleteMediaData`

8. In the `web.config` file, comment out the following section:

```
<database id="web" singleInstance="true" type="Sitecore.Data.Database,
Sitecore.Kernel">
<database>
```

Also, comment out any websites that use the Web database:

```
<site name="modules website">
  <site name="website">
```

To configure aggregation agents or threads on a processing server:

1. Open the `Sitecore.Analytics.Processing.Aggregation.config` file in an XML editor. You can specify how many aggregation agents you want to run at the same time. Start by using the default settings.

2. Edit the `<MaxThreads>` setting to specify the number of aggregation agents (threads) that you want to run on the server. You can also specify how many cleanup and recovery threads that you need. Adjust these settings depending on your available hardware and business requirements.

```
<!-- Aggregation Module: -->
<module type="Sitecore.Analytics.Aggregation.AggregationModule"
  singleInstance="true">
  <BackgroundServices hint="list:Add">
    <aggregator type="Sitecore.Analytics.Aggregation.BackgroundService">
      <param desc="agentName">aggregation/aggregator</param>
      <Interval>0.00:00:15</Interval>
      <MaxThreads>16</MaxThreads>
    </aggregator>
    <cleanup type="Sitecore.Analytics.Aggregation.BackgroundService">
      <param desc="agentName">aggregation/cleanup</param>
      <Interval>0.00:00:15</Interval>
      <MaxThreads>16</MaxThreads>
    </cleanup>
    <recovery type="Sitecore.Analytics.Aggregation.BackgroundService">
      <param desc="agentName">aggregation/recovery</param>
      <Interval>0.00:00:15</Interval>
      <MaxThreads>16</MaxThreads>
    </recovery>
    <history type="Sitecore.Analytics.Aggregation.BackgroundService">
      <param desc="agentName">aggregation/historyWorker</param>
      <Interval>0.00:00:15</Interval>
      <MaxThreads>16</MaxThreads>
    </history>
    <historyCompletionCheck
     type="Sitecore.Analytics.Aggregation.BackgroundService">
      <param desc="agentName">aggregation/historyCompletionCheck</param>
      <Interval>0.00:00:15</Interval>
      <MaxThreads>1</MaxThreads>
    </historyCompletionCheck>
  </BackgroundServices>
</module>
```

Start by using the default values and then over time adjust these settings to optimize your aggregation server or servers.

In this type of configuration, the processing server performs two functions: aggregation and processing. These are the two main roles for a processing server.

Aggregation Performance Tip

If you rebuild the entire MongoDB analytics database, then you might experience some performance issues during aggregation processing.

To optimize performance during aggregation, edit the Lucene and SOLR analytics configuration files and adjust the minimum and maximum queue size values.

To adjust the minimum and maximum queue size settings, open one of the following configuration files in the `Website\App_Config` folder:

- `Sitecore.ContentSearch.Lucene.Index.Analytics.config`
- `Sitecore.ContentSearch.Solr.Indexes.Analytics.config`

Note
The Solr indexing configuration file is only present here if you have installed Solr.

Increase the `MinimumQueueSize` value if your throughput is extremely high.

```
<crawler type="Sitecore.ContentSearch.Analytics.Crawlers.AnalyticsVisitPageCrawler,
 Sitecore.ContentSearch.Analytics">
                <CrawlerName>Lucene Visit Page Crawler</CrawlerName>
                <ObservableName>VisitPageObservable</ObservableName>
                <NumberOfSecondsToQueue>60</NumberOfSecondsToQueue>
                <MinimumQueueSize>500</MinimumQueueSize>
                <MaximumQueueSize>50000</MaximumQueueSize>
</crawler>
```

Note

Before running history processing using the `RebuildReportingDb.aspx` page, make sure you add a *reporting.secondary* connection string to the `ConnectionStrings.config` file.

For more information on enabling other features on a server, see Server Features.

Reporting Service Server

A Reporting Service server requires the *Reporting Service* server feature.

To configure a dedicated *Reporting Service* server:

1. Install Sitecore instance from Sitecore distributive.

2. Navigate to the `/App_Config/Include` folder and disable the following configuration files. To disable a configuration file change the extension from `.config` to `.disabled`:

   o `Sitecore.Analytics.Tracking.Database.config`

   o `Sitecore.Analytics.Tracking.config`

   o `Sitecore.Analytics.Tracking.Aggregation.config`

   o `Sitecore.Analytics.Processing.Aggregation.Services.config`

   o `Sitecore.Analytics.Processing.Services.config`

   o `Sitecore.Analytics.Automation.TimeoutProcessing.config`

   o `Sitecore.Analytics.Tracking.RobotDetection.config`

3. Navigate to the `/App_Config/Include` folder and enable the following configuration file:

   o `Sitecore.Analytics.Reporting.RemoteServer.config`

4. In the `/App_Config/ConnectionStrings.config` file comment out or remove the `tracking.live` and `tracking.history` connection strings:

   ```
   <add name="tracking.live " connectionString="mongodb:….."/>
   <add name="tracking.history" connectionString="mongodb:….."/>
   ```

In this type of configuration, the dedicated server performs a single function as the Reporting Service which can query multiple data sources such as the collection or reporting databases to gather data for

dashboards and reports. We recommend that you only configure one and not multiple dedicated Reporting Service servers.

For more information on enabling other features on a server, see *Server Features*.

To enable communication between the content management server (client) and the Remote Reporting Service, you must make the following configuration changes:

1. On your content management server client, in the `Website/App_Config` folder, open the `Sitecore.Analytics.Reporting.RemoteClient.config` file and enable it by removing `.disabled` from the file name.

2. Under the `<httpTransportFactory>` node, change the default value of the desc parameter to the hostname of your Reporting Service server instance.

   See the following example:

   ```
   <httpTransportFactory
           type="Sitecore.Analytics.Commons.ConfigurationBasedHttpTransportFactory,
           Sitecore.Analytics" singleInstance="true">
       <param desc="serviceUrl">http://reportingservice/</param>
   </httpTransportFactory>
   ```

3. In the `Website/App_Config` folder, open the `ConnectionStrings.config` file and comment out or remove the following reporting, tracking.live and tracking.history connection strings:

   ```
   <add name="reporting" connectionString="….."/>
   <add name="tracking.live " connectionString="mongodb:….."/>
   <add name="tracking.history" connectionString="mongodb:….."/>
   ```

Collection Database Server (xDB)

To configure a dedicated collection database server.

1. Install a blank MongoDB database by following the instructions on the MongoDB website.

   For hardware and software guidelines see, Hardware Guidelines and Software Recommendations.

   Note
   The xDB MongoDB analytics database is used as the xDB collection database. So when you create a dedicated collection database server there is no need to install a Sitecore instance.

2. Install the Windows version of MongoDB and ensure that it is running as a service.

3. Configure a MongoDB three-server replica set – To see an example of a standard MongoDB replica set architecture that consists of three MongoDB instances (primary, secondary and arbiter), see the *xDB Overview and Architecture document*.

   Note
   For production environments, we do not recommend that you configure a standalone MongoDB instance instead you should configure a replica set to ensure automatic failover and data safety.

4. Once you have configured the collection database on a dedicated server then all other servers that have an analytics connection string or need to connect to the MongoDB collection database should be configured to point to this server and to use the correct port number.

Session Database Server

You can install a session database on either a MongoDB or SQL Server dedicated server.

Note
A dedicated Sitecore instance is not required when configuring a dedicated session state database server.

To use MongoDB or SQL Server as dedicated session servers you also need to install the appropriate session state provider:

- Sitecore ASP.NET Session State Provider for MongoDB

- Sitecore ASP.NET Session State Store Provider for Microsoft SQL Server

Ensure that you configure connection strings on any other servers that need to connect to the session state database server.

For more information on configuring session state servers, see the section on Session State.

Reporting Database Server

A reporting database server does not need to be installed or run on a Sitecore instance. You can install a reporting database as a dedicated SQL Server instance.

To configure a Microsoft SQL Server instance as a dedicated reporting server:

1. Install SQL Server 2008 R2 SP1 or later.

2. On your content management and processing instances, open the *ConnectionStrings.config* file, modify the connection strings to point to the dedicated reporting database server.

3. Ensure that you configure connection strings for any other servers that need to connect to the reporting database server.

4. Ensure that your SQL Server instance is configured and running correctly.

5. Test your connections.

For more information on configuring connection strings for the reporting database, see section 2.4 Configuring Connection Strings.

Note
A SQL Server reporting can only be scaled vertically.

2.6 Session State

A session state server stores information relevant to current contact sessions. The Sitecore xDB comes with two session state services:

- **Private session state** holds information private to sessions. This is contact visit information, such as pages viewed, goals converted, or campaigns triggered.

- **Shared session state** holds information that may be shared by multiple visits on the same cluster, such as contacts and devices.

You can configure session state in two different ways; as either in process (`InProc`) or out of process.

Important

Sitecore requires that you must configure both private and shared session state. You can choose to use the same database for both private and shared sessions but to ensure that the system can distinguish between the private and shared session entries you must remember to configure the `sessionType` attribute in the `web.config` file correctly.

2.6.1 In Process

In process (`InProc`) is the default session state provider that comes with the Microsoft .NET Framework. It uses internal memory to track interactions and visits.

In process is the most suitable way of handling private session state for all data related to a specific interaction (single visitor session or visit). It is the recommended mode to use for content management servers.

Note

You cannot use `InProc` in a load balanced environment unless you configure the load balancer to use sticky sessions.

2.6.2 Out of Process

Out of process means that you use an external ASP.NET session state provider such as MongoDB or SQL Server. This is suitable for handling shared session state if you have multiple content delivery servers. It is also suitable if you have multiple content management servers and you do not wish to use sticky sessions.

Sitecore comes with the following two session state providers for configuring out of process session state:

- Sitecore ASP.NET Session State Provider for MongoDB

- Sitecore ASP.NET Session State Store Provider for Microsoft SQL Server

2.6.3 Session State Configuration Scenarios

The following example scenarios provide guidance on how to configure session state on several different types of Sitecore server configurations.

Single Sitecore Server

On a single standalone Sitecore instance both session providers should use the *in process* (`InProc`) session state mode.

Single Content Delivery Server and a Separate Content Management Server

On the content delivery server you should configure both session providers to use the *in process* (`InProc`) session state mode.

Important

When configuring session state on content management servers you should always configure session state providers to use the *in process* (`InProc`) mode. Sitecore xDB does not support sharing session state among a group of content management servers or between content management and content delivery servers.

Content Delivery Cluster with a Sticky Load Balancer

On all content delivery instances you should configure the ASP.NET session state provider to use the *in process* (`InProc`) mode. You should configure the shared session state provider to use one of the *out of process* modes connected to a database shared among all the content delivery instances.

Content Delivery Cluster with a Non-Sticky Load Balancer

On all content delivery instances you should configure both session state providers to use one of the *out of process* modes connected to a database (or databases) shared among the content delivery instances.

For an overview of session state, see the *xDB Overview and Architecture* document.

For more information on scalability settings and installing a dedicated content delivery server, see the *Sitecore Scaling Guide*.

2.7 Configuring Private Session State

Private session state holds visit information private to contact sessions such as pages viewed, goals converted, campaigns triggered or engagement points accumulated.

You can use either MongoDB or SQL Server as your private session state store. If you are running an *on-premise* solution with a MongoDB database as your collection database then we recommend that you use MongoDB as your session store.

The *Sitecore ASP.NET Session State Provider for MongoDB* allows you to use MongoDB as your session state store. This provider supports the `SessionEnd` event, which the xDB needs to track website visits.

Note
Private session state is not required to support the `Session_End` event on content management servers. Private session state must support the `Session_End` event on content delivery servers.

Follow the steps in this section to configure a MongoDB or SQL Server session state provider.

Note
If you are using a MongoDB session provider then all content delivery servers should use the same provider pointing to the same database.

2.7.1 Configuring the MongoDB Session State Provider

Follow the steps in this section to use a MongoDB database as your private session state store using the *Sitecore ASP.NET Session State Provider for MongoDB*.

MongoDB Hardware Considerations

For each web request the content delivery server accesses the session state store database multiple times. This can have a significant impact on the performance of your web site, so we recommend that you use solid-state drives for the database and install enough memory to avoid frequent disk reads.

Each visit requires about 30 kilobytes of storage capacity in the session database (by default) but space requirements may change depending on customizations.

You can calculate the disk space requirements expressed in kilobytes using the following formula:

- Maximum number of concurrent visits * 30

Note
This includes active visits and those which are inactive but which have not yet timed out.

You can also use the same formula to calculate memory requirements.

See the following MongoDB documents for more details:

http://docs.mongodb.org/manual/faq/diagnostics/#how-do-i-calculate-how-much-ram-i-need-for-my-application

http://docs.mongodb.org/manual/faq/storage/#what-is-the-working-set

For more information on how to install and configure a MongoDB database, see the steps outlined earlier in section 2.3.3 Install MongoDB and the MongoDB website.

Deploying the MongoDB Session Database

To configure the *Sitecore ASP.NET Session State Provider for MongoDB*:

1. Install MongoDB database server version 2.6 or later. We recommend that you install this on a dedicated server.

2. Configure Sitecore to use the *Sitecore ASP.NET Session State Provider for MongoDB* (see the instructions in this section).

Configuring Sitecore

Configure the ASP.NET session state provider for MongoDB in the same way as any other custom session state store provider.

To connect the MongoDB session provider:

1. Open the `ConnectionStrings.config` file located here
   `<sitename>\Website\App_Config` and add the following connection string:

   ```
   <add name="session"
   connectionString="mongodb://_mongo_server_name_:_port_number_/_session_database_name_"
   />
   ```

2. Open the `web.config` file in your site root folder `<sitename>\Website` and locate the `sessionState` section:

   ```
   <sessionState mode="InProc" cookieless="false" timeout="20">
   ```

3. Update the `sessionState` section to use the MongoDB provider instead of `InProc` as shown in the following example. Also, change the `name` attribute value to `mongo`:

   ```
   <sessionState mode="Custom" customProvider="mongo" cookieless="false" timeout="20">
     <providers>
       <add name="mongo"
           type="Sitecore.SessionProvider.MongoDB.MongoSessionStateProvider,
           Sitecore.SessionProvider.MongoDB"
           connectionStringName="session"
           pollingInterval="2"
           compression="true"
           sessionType="private"/>
     </providers>
   </sessionState>
   ```

   Configuration options:

Setting	Description

`connectionStringName`
 	
For example:
`connectionStringName="session"`
	Edit the connection string so that the session provider connects to the SQL Server database that you want to use. In the xDB this database is called `session`.

`Polling interval`
 	
For example:
`pollingInterval="2"`
	This is the time interval in seconds that the session store provider uses to check if any sessions have expired.

`Compression`
 	
For example:
`compression="true" />`
	This setting is a Boolean flag that indicates whether to compress session state data or not. The default value is *true*. Compressing session state data reduces the amount of data that you need to transfer between the database and the Sitecore instance. This may cause some additional CPU overhead.

`sessionType`
 	
For example:
`sessionType="private"`
	This value must be set either to *private* or *shared*.

If you have configured everything correctly, a `session` database should appear in your list of MongoDB databases after the first web request.

2.7.2 Configuring the SQL Server Session State Provider

Follow the steps in this section to use a SQL Server database as your private session state store using the *Sitecore ASP.NET Session State Provider for SQL Server.*

This might be an appropriate option if you are running the collection database (MongoDB) in the cloud as a service or if you prefer not to run an on-premise MongoDB server instance.

This provider supports the session end event which is required by the xDB in order to track website visits.

To configure Private Session State Provider for Microsoft SQL Server:

- In Microsoft SQL Server, deploy the *Session* database
- Configure Sitecore to use the SQL Server session provider.

Deploying the SQL Server Session Database

For each web request the session state store database is accessed multiple times. This can have a significant impact on the performance of your web site. Therefore we recommend that you install enough RAM to allow Microsoft SQL Server to keep the session state database in memory. We also recommended that you put the database files on an SSD drive.

To deploy the *Session* database:

1. Start Microsoft SQL Server Management Studio 2012 or later.
2. Connect to the server node that you want to install the Session database on.
3. Expand the server node, right-click **Databases,** and then click **Attach**.
4. In the **Attach Databases** dialog box, click **Add**.
5. Browse to the **Databases** folder in your website root folder, select the `Sitecore.Sessions.mdf` database and click **OK**.
6. In the **Attach Databases** dialog box, click **OK**. The session database now appears in your list of attached databases.
7. Add the following connection string to the `ConnectionStrings.config` file:

```
<add name="sharedsession" connectionString="user
id= sql server user ;password= user password ;Data
Source=_sqlserver_;Database = _sharedSession_database_name_"/>
```

The `add name` value can be `session` or `sharedsession` depending on whether you are configuring private or shared session state.

Performance Optimizations

To achieve optimal performance you can install an extension to the *Sessions* database.

To install the performance enhancements start *Microsoft SQL Server Management Studio 2012*:

1. Open the `Performance Boost.sql` file.
2. In the first line of the `Performance Boost.sql` file replace `USE [Sitecore_Session]` with the name of your session database.
3. After you have updated the USE statement to point to your session database hit F5 key to execute the file.

Note
The performance enhancements are not supported on Windows Azure.

Configuring Sitecore

You configure the ASP.NET session state store provider for Microsoft SQL Server in the same way as any other custom session state store provider.

To connect to the SQL Server session provider:

1. Open the `ConnectionStrings.config` file located here `<sitename>\Website\App_Config` and add the following connection string:

```
<add name="session" connectionString="user
id=_sql_server_user_;password=_user_password_;Data
Source=_sqlserver_;Database=_session_database_name_" />
```

2. Open the `web.config` file in your site root folder `<sitename>\Website` and locate the `sessionState` section:

```
<sessionState mode="Custom" customProvider="mssql" cookieless="false" timeout="20">
```

3. Update the `sessionState` section by adding the SQL Server provider as shown in the following example. Also, change the `name` attribute value to `mssql`:

```
<sessionState mode="Custom" customProvider="mssql" cookieless="false" timeout="20">
  <providers>
    <add name="mssql"
         type=" Sitecore.SessionProvider.Sql.SqlSessionStateProvider,
         Sitecore.SessionProvider.Sql"
         connectionStringName="session"
         pollingInterval="2"
         compression="true"
         sessionType="private"/>
  </providers>
</sessionState>
```

Configuration options:

Setting	Description

`connectionStringName`
 	
For example:
`connectionStringName="session"`
	Edit the connection string so that the session provider connects to the SQL Server database that you want to use. In the xDB this database is called `session`.

`Polling interval`
 	
For example:
`pollingInterval="2"`
	This is the time interval in seconds that the session store provider uses to check if any sessions have expired.

`Compression`
 	
For example:
`compression="true" />`
	This setting is a Boolean flag that indicates whether to compress session state data or not. The default value is *true* Compressing session state data reduces the amount of data that you need to transfer between the database and the Sitecore instance. This may cause some additional CPU overhead.

`sessionType`
 	
For example:
`sessionType="private"`
	This value must be set either to *private* or *shared*.

2.8 Configuring Shared Session State

The shared session state store holds data that can be shared by multiple sessions, such as data related to contacts and devices.

You must always configure shared session state whatever your server configuration. For example, you could have a single standalone content delivery server, multiple content delivery servers, or a clustered environment but you always need to configure session state.

Note
Shared session state is not supported on content management servers. On content delivery servers shared session state must support the `SessionEnd` event.

A contact can make multiple parallel visits to a web site in which case each visit will have its own private session state. However, some data may be shared between visits such as device and contact related information.

Information that may be shared between parallel visits of the same contact is stored in shared session state. This data is still private to the contact but it is accessible from all current sessions made by the same contact.

You can use either the Sitecore MongoDB or Sitecore SQL Server providers to configure your shared session state store. Both these providers support the `SessionEnd` event, which the xDB needs to track website visits.

Note
The standard SQL Server session state provider that is shipped with ASP.NET does not support the SessionEnd event so cannot be used with the xDB.

You can configure shared session state to use any session state store provider that extends the abstract class `SessionStateStoreProviderBase` (shipped with ASP.NET). The only additional requirement is that the session state store provider can invoke the `SessionEnd` event via `SessionStateItemExpireCallback`.

2.8.1 Configuring the MongoDB Session State Provider

Follow the steps in this section to use a MongoDB database as your shared session state store using the *Sitecore ASP.NET Session State Provider for MongoDB*.

Deploying the MongoDB Session Database

To configure the *Sitecore ASP.NET Session State Provider for MongoDB*:

1. Install MongoDB database server version 2.6 or later. We recommend that you install this on a dedicated server.

2. Configure Sitecore to use the *Sitecore ASP.NET Session State Provider for MongoDB* (see the instructions in this section).

Configuring Sitecore

To configure the Shared Session State Provider for MongoDB:

1. Open the `ConnectionStrings.config` file located here
   `<sitename>\Website\App_Config` and add the following connection string:

   ```
   <add name="sharedsession"
   connectionString="mongodb://_mongo_server_name:_port_number/_session_database_name" />
   ```

2. In your website root folder, navigate to:

   `Website\App_Config\Include`

3. Open the `Sitecore.Analytics.Tracking.config` file.

4. Locate the line where you have defined the default shared session state provider. Navigate to the following path: `sitecore/tracking/sharedSessionState`.

5. The default shared session store uses `inProc` provider (storing data in memory and implemented in the internal ASP.NET class `InProcSessionStateStore`):

```
<sharedSessionState defaultProvider="inProc">
  <providers>
    <clear/>
    <add name="inProc" type="System.Web.SessionState.InProcSessionStateStore" />
  </providers>
```

Note

For clustered environments, Sitecore ships with a MongoDB session state store provider.

6. In `Sitecore.Analytics.Tracking.config` file, update the `sessionState` section as follows

To configure MongoDB as your shared session state store provider change the `defaultProvider` from `inProc` to `mongo`. Also, change the `name` attribute value to `mongo`.

```
<sharedSessionState defaultProvider="mongo">
  <providers>
    <clear/>
    <add
      name="mongo"
      type="Sitecore.SessionProvider.MongoDB.MongoSessionStateProvider,
      Sitecore.SessionProvider.MongoDB"
      connectionStringName="sharedsession"
      pollingInterval="2"
      compression="true"
      sessionType="shared"/>
  </providers>
```

Configuration options:

Setting	Description

`connectionStringName`
 	
For example:
`connectionStringName="sharedsession"`
	Edit the connection string so that the session provider connects to the MongoDB database that you want to use. In the xDB this database is called `session`.

`Polling interval`
 	
For example:
`pollingInterval="2"`
	This is the time interval in seconds that the session store provider uses to check if any sessions have expired.

`Compression`
 	
For example:
`compression="true" />`
	This setting is a Boolean flag that indicates whether to compress session state data or not. The default value is *true*. Compressing session state data reduces the amount of data that you need to transfer between the database and the Sitecore instance. This may cause some additional CPU overhead.

`sessionType`
 	
For example:
`sessionType="shared"`
	This value must be set either to *private* or *shared*.

2.8.2 Configuring the SQL Server Session State Provider

Follow the steps in this section to use a SQL Server database as your shared session state store using the *Sitecore ASP.NET Session State Provider for SQL Server*.

Deploying the SQL Server Session Database

1. Use Microsoft SQL Server Management Studio 2012 or later to deploy your shared session store database. For more detailed instructions, see Deploying the SQL Server Session Database.

2. Provide an appropriate name for your session database. For example, `sharedsession`.

Configuring Sitecore

To configure the shared session state provider for SQL Server:

1. Add the following connection string to `ConnectionStrings.config`:

    ```
    <add name="sharedsession" connectionString="user
    id=_sql_server_user_;password=_user_password_;Data
    Source=_sqlserver_;Database = _sharedSession_database_name_"/>
    ```

2. In your website root folder, navigate to:

    `Website\App_Config\Include`

3. Open the `Sitecore.Analytics.Tracking.config` file.

4. Locate the line where you have defined the default shared session state provider. Navigate to the following path: `sitecore/tracking/sharedSessionState`.

5. To configure SQL Server as your shared session state store provider change the `defaultProvider` from `inProc` to `mssql`. Also, change the `name` attribute value to `mssql`.

    ```
            <sharedSessionState defaultProvider="mssql">
              <providers>
                 <clear/>
                 <add
        name="mssql"
        type="Sitecore.SessionProvider.Sql.SqlSessionStateProvider,Sitecore.SessionProv
    ider.Sql"
        connectionStringName="sharedsession"
        pollingInterval="2"
        compression="true" sessionType="shared"/>
                    </providers>
    ```

Configuration options:

Setting	Description

`connectionStringName`
 	
For example:
`connectionStringName="sharedsession"`
	Edit the connection string so that the session provider connects to the SQL Server database that you want to use. In the xDB this database is called `sharedsession`.

`Polling interval`
 	
For example:
`pollingInterval="2"`
	This is the time interval in seconds that the session store provider uses to check if any sessions have expired.

| Setting	Description

`Compression`
 	
For example:
`compression="true" />`
	This setting is a Boolean flag that indicates whether to compress session state data or not. The default value is *true* Compressing session state data reduces the amount of data that you need to transfer between the database and the Sitecore instance. This may cause some additional CPU overhead.

`sessionType`
 	
For example:
`sessionType="shared"`
	This value must be set either to *private* or *shared*.

For more information on session state, see the xDB *Overview and Architecture* document.

2.9 Clustered Environment Overview

A clustered environment consists of a collection of dedicated servers grouped together to improve scalability. For example, you could create multiple content delivery clusters to enable horizontal scaling with the aim of increasing storage capacity and improving performance. Each cluster could contain two or more content delivery instances, each with its own dedicated session state server. You could also group clusters together in the same location or spread them across different geographical locations.

In a clustered environment, the same cluster of web servers should serve all the visits for a single contact. This ensures that all visits have fast access to the current state of the contact. If a single contact opens multiple concurrent sessions from different browsers or devices then the xDB ensures that each session sticks to the same cluster. A contact can only move to another cluster once they end all their current open sessions and start a new one or when the session expires.

To ensure that contacts stick to the same cluster (as mentioned above) each cluster should have a shared session state server, as shown in the following diagram.
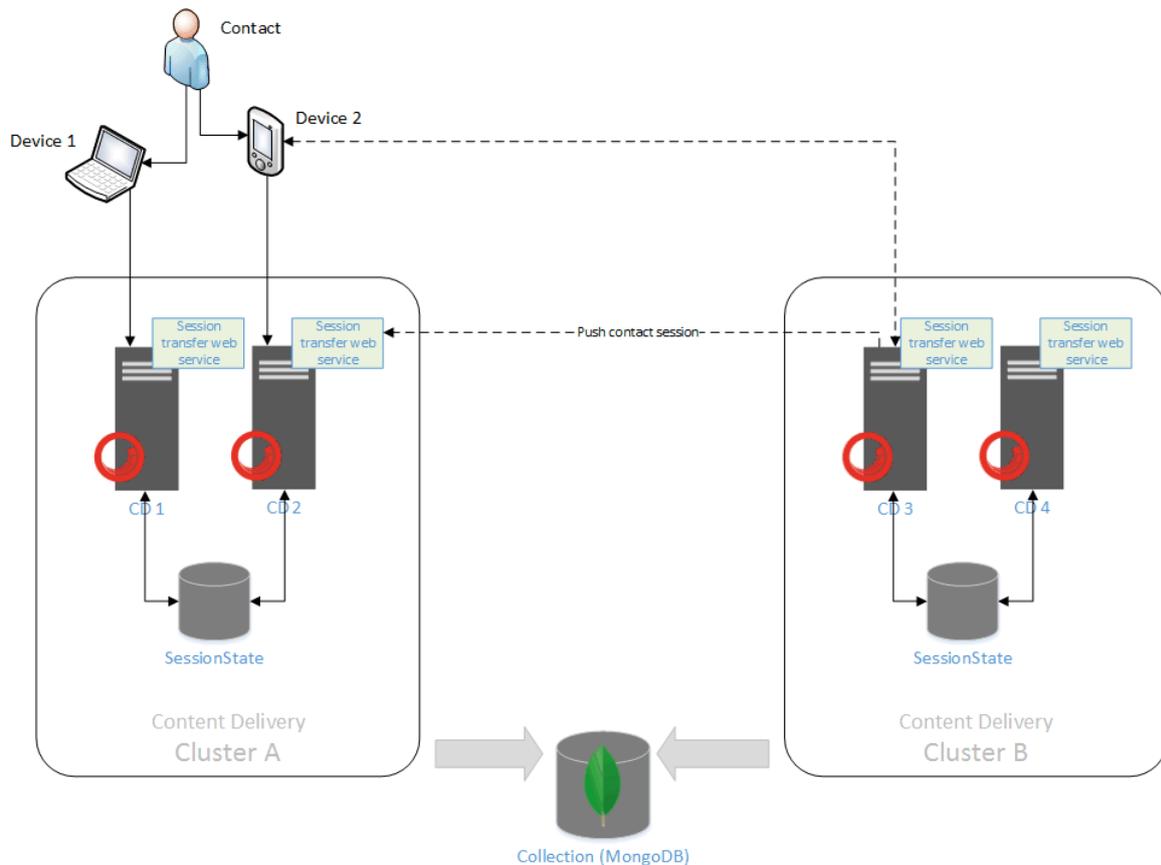
In this diagram, *Device 1* is the first device the contact uses to connect to the website. This session is still active when the same contact begins a second parallel visit using *Device 2*.

When a contact record is loaded into the shared session state of a cluster, the collection database locks it and allocates it to the current cluster. This ensures that a contact record is not loaded into two clusters at the same time.

For more detailed information on creating clustered environments, see the Sitecore scaling documentation on SDN.

Chapter 3

Cloud Configuration Options

You can deploy Sitecore partly or entirely in the cloud. This chapter outlines some of the advantages and disadvantages of using the cloud and how you should approach making your first cloud deployment.

This chapter contains the following sections:

- Cloud Server Solutions

3.1 Cloud Server Solutions

You can choose to install Sitecore xDB as an entirely on-premise solution, a partly on-premise solution, partly in the cloud (hybrid) or entirely in the cloud.

Sitecore xDB Cloud Edition is a service that enables you to run Sitecore xDB entirely in the cloud. This includes the following:

- Sitecore application servers for processing, aggregation and reporting.

- Microsoft SQL Server reporting database

- MongoDB collection database

The following table summarizes the on-premise and cloud solution options for Sitecore xDB and how they can be combined:

Available	xDB on-premises	xDB Cloud Edition
Sitecore CEP 7.2 (on-premises)	No	No
Sitecore® XP 7.5 (on-premises)	Yes	Yes
Sitecore® XP 7.5 (deployed using Sitecore Azure 7.5)	No	Yes

There are several other cloud offerings currently available to choose from, each with their own set of advantages and disadvantages. Think carefully before choosing a cloud platform other than Sitecore. Other current offerings include *Rackspace*, and *Amazon Web Services (AWS)* that both offer public cloud computing services for a fee.

Benefits of using Sitecore xDB Cloud Edition:

- Powerful, scalable, and fully managed – saves the cost of acquiring, configuring, and maintaining the infrastructure to support xDB on the premises.

- Easy to connect to, simple to set up and use.

- Low cost.

- Highly available and backed by SLA guarantees.

- Globally available.

- Easy to increase the number of servers.

- Available for production and non-production usage. Non-production instances are provided for development and testing.

For more information on using the Sitecore xDB Cloud Edition, see *xDB Cloud Quick Start Guide* on SDN or contact your local Sitecore office.

Chapter 4

Customization

This chapter explains the steps you need to follow to create and implement a custom aggregation pipeline.

This chapter includes the following sections:

- Creating a Custom Aggregation

- Fact and Dimension Tables

- Creating Dimension Tables

- Implementing Model Classes

- Implementing the Aggregation Processor

- Example Custom Aggregation

4.1 Creating a Custom Aggregation

In the xDB, aggregation describes a type of processing that reduces and adapts data from the collection database (MongoDB) so that it can be stored in the reporting database (SQL Server). Aggregated collection data is then made available to reporting applications such as the *Executive Insight Dashboard* and *Engagement Analytics* reports. Ideally, the aggregation server should be a dedicated Sitecore CMS server with the client removed.

You can create your own custom aggregations by extending the aggregation pipeline, for example you could create a custom aggregation for website visitors to rate blog posts by adding their own star ratings. This example is described in Section 4.5 Example Custom Aggregation.

The purpose of this document is to describe the general steps required to implement a new aggregation processor.

Summary of steps:

- Create the required database tables

- Create the stored procedures (optional)

- Implement the model classes

- Implement and configure the pipeline

- Implement a script builder (optional)

4.2 Fact and Dimension Tables

Fact tables contain the measurements and metrics of the process you are monitoring.

Metrics could be the engagement value accumulated during a visit or the number of events generated during a page request. When creating reports, it is often necessary to investigate and compare facts with different attributes. For example, compare the performance of a campaign over the last few months with its current performance. The attributes used in this example are the months and the campaign. These allow us to filter out the two sets of facts we want to compare with each other.

If the storage requirements of an attribute are large or it is repeated for many visits, we may choose to put it into a separate table. These separate tables are referred to as dimensions.

There is a fact table for each report, and each fact table can have zero or more dimension tables connected to it. This means that if you create a new report, you typically need to create a new fact table to support it. See the section Creating Fact Tables for the script you need to create fact tables.

Dimension tables can be shared among fact tables. This means that if two fact tables include the same Item ID attribute, they can both refer to the same Items dimension.

Decide if you need to create any new dimension tables and if so, first check whether they are already present. The most common dimension tables are already defined in the reporting database along with the fact tables. The stored procedures and the model classes for these may also be present already.

The most common dimension tables are:

- Site Names
- Device Names
- Languages
- Campaigns
- Items
- Keywords
- Referring Sites

When you have decided which fact and dimension tables you need for your custom aggregations, then create a script for each fact and dimension table which should be placed in its own `.sql` file.

4.2.1 Creating Fact Tables

The following SQL script shows how to define a fact table:

```sql
CREATE TABLE [Fact_Name]
(
  [K1] DATATYPE NOT NULL,
  [K2] DATATYPE NOT NULL,
  [Kn] DATATYPE NOT NULL,

  [V1] DATATYPE NOT NULL,
  [V2] DATATYPE NOT NULL,
  [Vn] DATATYPE NOT NULL,

  CONSTRAINT [PK_Fact_Name] PRIMARY KEY CLUSTERED ([K1], [K2], [Kn]),

  CONSTRAINT [FK_Fact_Name_K1] FOREIGN KEY ([K1]) REFERENCES [DimensionName] ([K1]),
  CONSTRAINT [FK_Fact_Name_Kn] FOREIGN KEY ([Kn]) REFERENCES [DimensionName] ([K2])
);

ALTER TABLE [Fact_TableName] NOCHECK CONSTRAINT [FK_Fact_Name_K1];
ALTER TABLE [Fact_TableName] NOCHECK CONSTRAINT [FK_Fact_Name_Kn];

CREATE NONCLUSTERED INDEX [IX_ByK1] ON [Fact_TableName]( [K1] );
```

We recommend that you create all foreign key constraints to make dependencies between tables visible, but we also recommend that you disable them to improve performance.

Note
Microsoft SQL Server has a limitation of 900 bytes for indices, so if the storage requirements of the data in the key columns exceeds 900 bytes, no primary key can be defined on the table. This is not a serious problem for fact tables, but it can affect performance under certain conditions. A possible solution is to extract parts of the key into dimension tables.

Note
A primary key may be clustered or non-clustered. A clustered index defines the physical order of rows in the table. If no physical order is defined, the table is referred to as a heap. The order in which the columns are listed in the constraint does matter. As a rule of thumb, put the column with the highest selectivity first and the one with the lowest selectivity last.

4.2.2 Creating Dimension Tables

Dimension tables are usually simpler than fact tables.

Use the following template for creating dimension tables:

```sql
CREATE TABLE [DimensionName]
(
  [K1] DATATYPE NOT NULL,
  [Kn] DATATYPE NOT NULL,

  [A1] DATATYPE NOT NULL,
  [An] DATATYPE NOT NULL,

  CONSTRAINT [PK_DimensionName] PRIMARY KEY CLUSTERED ( [K1], [Kn] )
);
```

Note
Dimension tables most often contain simple key-value pairs with a single key column and a single attribute column.

4.3 Implementing Model Classes

Both fact and dimension tables consist of one or more key values and one or more aggregated values. The diagram below shows the classes along with the base classes that represent the in memory model for the Traffic fact and the Items dimension.
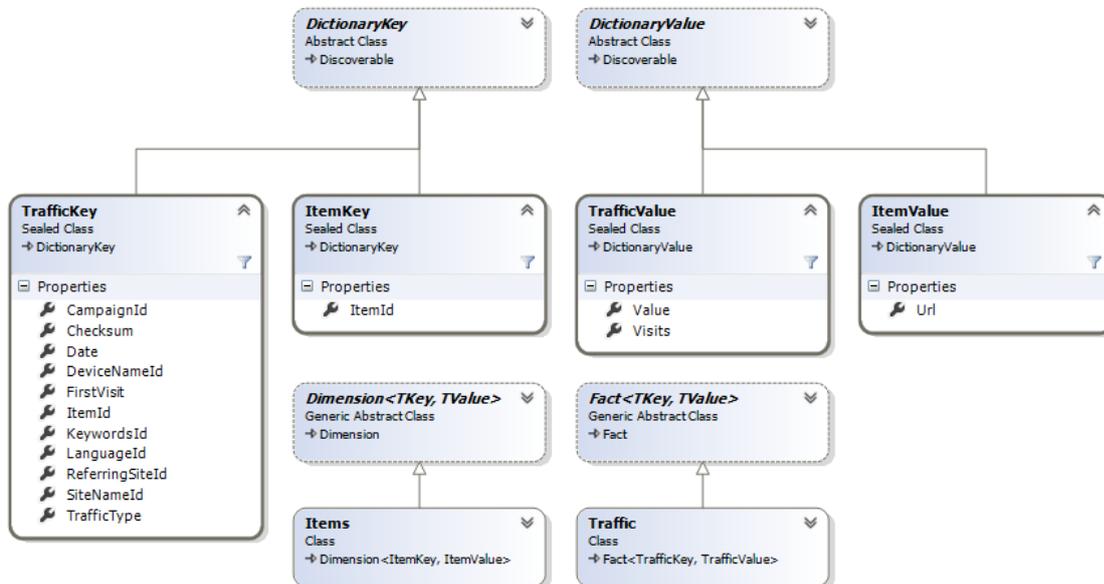
Implementing a Dimension Model Class

To implement a model class for a new dimension:

1. Create a new class that inherits from the `Sitecore.Analytics.Aggregation.Data.Model.DictionaryKey` class (`Sitecore.Analytics.Aggregation.dll`).

2. Expose all key fields as public properties (get and set).

3. Create a new class that inherits from the `Sitecore.Analytics.Aggregation.Data.Model.DictionaryValue` class (`Sitecore.Analytics.Aggregation.dll`).

4. Add the required properties.

5. Both the key and the value class must have a public default constructor.

6. Create a new class that inherits from the `Sitecore.Analytics.Aggregation.Data.Model.Dimension<TKey, TValue>` class (`Sitecore.Analytics.Aggregation.dll`), with a public default constructor.

Implementing a Fact Model Class

Implementing the class model for a fact is very similar to implementing the class model for a dimension.

To implement a model class for a new fact:

1. Create a new class that inherits from the `DictionaryKey` class.

2. Expose all key fields as public properties (get and set).

3. Create a new class that inherits from the `DictionaryValue` class.

4. Add the required properties.

5. Both the key and the value class must have a public default constructor.

---

6. Create a new class that inherits from the
   `Sitecore.Analytics.Aggregation.Data.Model.Fact<TKey, TValue>` class
   (`Sitecore.Analytics.Aggregation.dll`), with a public default constructor.

   The constructor of the base class takes a delegate as the one and only argument. This
   method is called whenever two values need to be aggregated.

   Sample implementation:

   ```
   internal static TrafficValue Reduce(TrafficValue left, TrafficValue right)
   {
    TrafficValue result = new TrafficValue();

    result.Visits = (left.Visits + right.Visits);
    result.Value = (left.Value + right.Value);

    return result;
   }
   ```

   Note

   All property types exposed by the key class or the value class need to be serializable. If they
   are not then you need to override the `GetHashCode()` and `Equals()` methods.

4.4 Implementing the Aggregation Processor

Follow these steps to implement the aggregation processor:

1. Create a class that inherits from the
   `Sitecore.Analytics.Aggregation.Pipeline.AggregationProcessor` base class
   (`Sitecore.Analytics.Aggregation.dll`).

2. Override the `Process(AggregationPipelineArgs args)` method. The `args` parameter
   provides access to the interaction to be processed and to all dimensions and all facts.

   The interaction can be accessed via the `Context` property. Dimensions and facts can be
   accessed via the following methods:

   - `MyFact fact = args.GetFact<MyFact>();`

   - `MyDimension dimension = args.GetDimension<MyDimension>();`

3. Implement the logic that extracts the information from the aggregation context (the visit in the
   example above) and populate the fact and dimensions.

4. Register your processor in the
   `Sitecore.Analytics.Processing.Aggregation.config` file.

   The following example shows where you add your own processor in the aggregation
   configuration file:

```xml
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>

    <pipelines>

      <group groupName="analytics.aggregation">
        <pipelines>

          <interactions>
              <processor type="MyNamespace.MyProcessor, MyAssembly" />
          </interactions>

        </pipelines>
      </group>

    </pipelines>

  </sitecore>
</configuration>
```

4.5 Example Custom Aggregation

There are several aggregations that come with Sitecore and that work *out of the box*. These are used by the *Executive Insight Dashboard* and other components in Sitecore.

If you add new functionality to your website which requires you to extract and present data from the xDB in a different way, then you can write your own custom aggregation.

Note
We recommend that you follow this approach if the data that you want to extract is from a high number of interactions or if the extraction process is relatively resource intensive.

4.5.1 Overview

In this chapter you will learn how to create a custom aggregation using an example developed for the *sitecore.net* website.

On the *sitecore.net* website, visitors can rate blog posts by adding their own star ratings. Rating data can then be used to sort and search through posts.
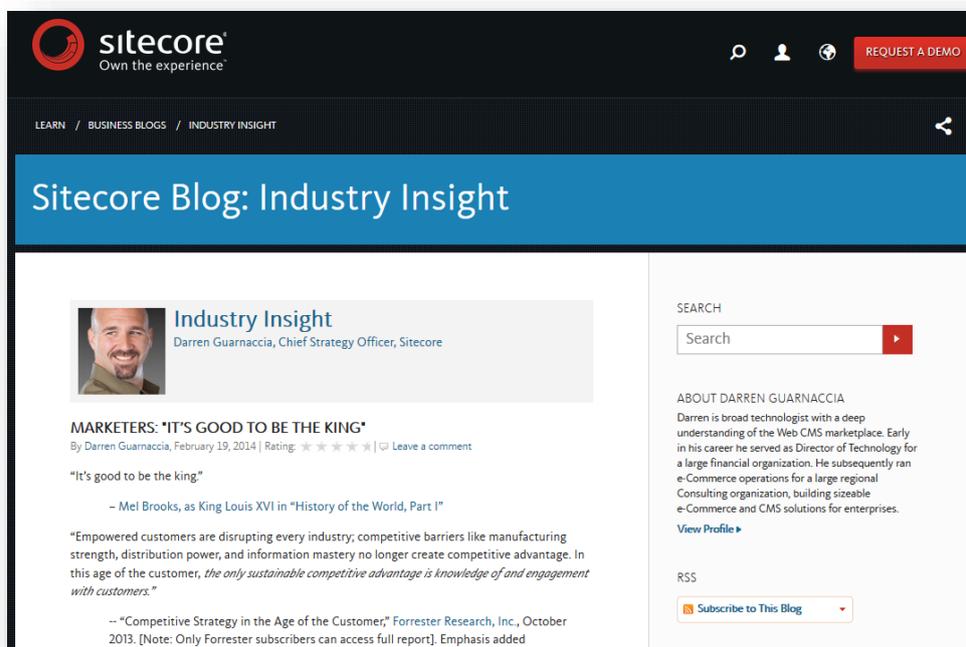
After you have implemented this custom aggregation you can add the additional option to search blogs by popularity.
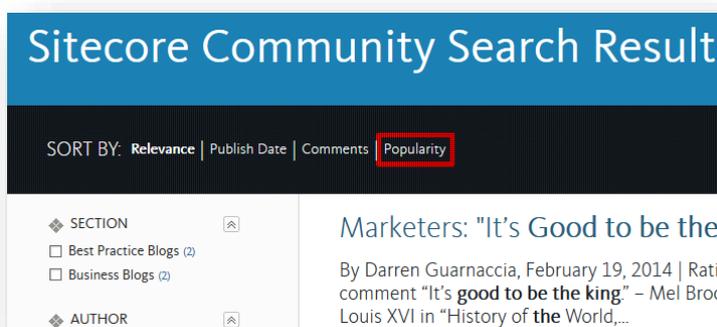
This chapter includes all the steps you need to follow and sample code that you can re-use in your own solution.

Summary of steps to follow:

1. Create a rate page event.

2. Create a new fact table.

3. Implement the model classes.

4. Implement the aggregation processor.

5. Register your custom aggregation in the aggregation configuration file.

4.5.2   Create a Rate Page Event

When a visitor clicks a star to rate the blog post, we want the system to register a new `Rate page` event for the current page. The selected value (from 1 to 5) is written to the `CustomValues` collection with key `DataCode`.

Example `Rate page` event using the Sitecore "Industry insight" blog post:

```
var eventData = new PageEventData("Rate")
            {
                Data = "Industry Insight",
                DataKey = "{7F84F941-A284-4DF5-9DE9-1B22A387039E}",
                Text = "Rated"
            };

var eventRow = Tracker.Current.Interaction.PreviousPage.Register(eventData);
eventRow.CustomValues["RateValue"] = 5;
```

4.5.3   Create a Fact Table

For this custom aggregation you need to create one new fact table. Fact tables contain the measurements of the experience data that you want to aggregate.

The following SQL script defines a fact table for this example:

```
CREATE TABLE [dbo].[Fact_Rating](
[ItemId]  UNIQUEIDENTIFIER NOT NULL,
[Rating]   BIGINT NOT NULL,
[Count]   BIGINT NOT NULL,
CONSTRAINT [PK_Rating] PRIMARY KEY CLUSTERED ([ItemId])
)
GO

ALTER TABLE [dbo].[Fact_Rating]  WITH NOCHECK
ADD   CONSTRAINT [FK_Fact_Rating_Items]
FOREIGN KEY([ItemId])
REFERENCES [dbo].[Items] ([ItemId])
GO

ALTER TABLE [dbo].[Fact_Rating] NOCHECK
CONSTRAINT [FK_Fact_Rating_Items]
GO
```

Where:

- `ItemId` – blog post ID(item ID in content tree)

- `Rating` – the total value given by all visitors

- `Count` – number of visitors who rated the post

4.5.4 Create Fact and Dimension Model Classes

To implement a class model for a new dimension, create a new class that inherits from the `DictionaryKey` class and expose all key fields as public properties (get and set).

`RatingKey.cs`

```csharp
namespace Sitecore.Components.Community.BusinessLayer
{
  using System;

  using Sitecore.Analytics.Aggregation.Data.Model;

  public class RatingKey : DictionaryKey
  {
    public Guid ItemId { get; set; }
  }
}
```

Create a new class that inherits from the `DictionaryValue` class and add the required properties. Both the key and the value class must have a public default constructor. Create a new class that inherits from the Dimension`<TKey, TValue>` class, again with a public default constructor.

`RatingValue.cs`

```csharp
namespace Sitecore.Components.Community.BusinessLayer
{
  using Sitecore.Analytics.Aggregation.Data.Model;

  public class RatingValue : DictionaryValue
  {
    internal static RatingValue Reduce(RatingValue left, RatingValue right)
    {
      var ratingValue = new RatingValue();

      ratingValue.Count = left.Count + right.Count;
      ratingValue.Rating = left.Rating + right.Rating;

      return ratingValue;
    }

    public long Rating { get; set; }

    public long Count { get; set; }
  }
}
```

`RatingFact.cs`

```csharp
namespace Sitecore.Components.Community.BusinessLayer
{
  using Sitecore.Analytics.Aggregation.Data.Model;

  public class Rating : Fact<RatingKey, RatingValue>
  {
    public Rating() : base(RatingValue.Reduce)
    {
    }

  }
}
```

4.5.5 Implementing the Aggregation Processor

Create a class that inherits from the `AggregationProcessor` base class. Implement the `OnProcess(AggregationPipelineArgs args)` method that is defined as an abstract method in the base class.

`AggregationProcessor.cs`

```csharp
namespace Sitecore.Components.Community.BusinessLayer
{
```

```csharp
using System;
using System.Linq;

using Sitecore.Analytics.Aggregation.Pipeline;
using Sitecore.Analytics.Model;
using Sitecore.Diagnostics;

public class RatingProcessor : AggregationProcessor
{
  private Guid RatePageEventDefinitionId = Guid.Parse("08E50AA5-E5E7-4845-940B-
2E81FB3ED56C");

    protected override void OnProcess(AggregationPipelineArgs args)
    {
      Assert.ArgumentNotNull(args, "args");

      if (args.Context.Visit.Pages == null)
      {
        return;
      }

      foreach (PageData page in args.Context.Visit.Pages)
      {
        if (page.PageEvents != null)
        {
          var fact = args.GetFact<Rating>();
          foreach (var pageEvent in page.PageEvents.Where(p => p.PageEventDefinitionId
          == RatePageEventDefinitionId))
          {
            int rating = GetRating(pageEvent);
            var postId = GetPostId(pageEvent);

            if (rating > 0 && postId != Guid.Empty)
            {
              var ratingKey = new RatingKey();
              ratingKey.ItemId = postId;

              var ratingValue = new RatingValue();
              ratingValue.Count = 1;
              ratingValue.Rating = rating;

              fact.Emit(ratingKey, ratingValue);
            }
          }
        }
      }
    }

    private int GetRating(PageEventData pageEvent)
    {
      int rating = 0;
      if (pageEvent.CustomValues.ContainsKey("RateValue"))
      {
        rating = (int)pageEvent.CustomValues["RateValue"];
      }

      return rating;
    }

    private Guid GetPostId(PageEventData pageEvent)
    {
      if (!string.IsNullOrEmpty(pageEvent.DataKey))
      {
        Guid postId;
        if (Guid.TryParse(pageEvent.DataKey, out postId))
        {
          return postId;
        }
      }

      return Guid.Empty;
    }
  }
}
```

4.5.6　Registering your Custom Aggregation

To ensure that your processor is actually executed, you need to add it to a configuration file and save it to the `App_Config/Include` folder. Create a new configuration file for this purpose called `Rating.Aggregation.config`.

The following example illustrates where you add you custom aggregation in the `Rating.Aggregation.config` file:

```xml
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>

    <pipelines>

      <group groupName="analytics.aggregation">
        <pipelines>

          <interactions>
              <processor
type="Sitecore.Components.Community.BusinessLayer.RatingProcessor,
Sitecore.Components.Community" />
          </interactions>

        </pipelines>
      </group>

    </pipelines>

  </sitecore>
</configuration>
```

Note
When you try out the example remember to update your type and assembly names if they are not using the same namespace or assembly.