Sitecore E-Commerce Services 1.2

Scaling Guide

How to run SES across multiple Sitecore instances

Introduction

This document describes how to run SES across multiple Sitecore instances.

You may need to run SES on multiple Sitecore instances for one or more of the following reasons:

- Scalability — you might need to have multiple content delivery (CD) servers for handling web-requests and multiple content management (CM) servers for managing the content as well as the products and orders.
- Separation — you might need to use a firewall to separate the CD servers from the CM servers. You can place the content management servers behind the firewall where they won't steal resources from the content delivery servers that are managing the web-requests.
- Security — managing orders, products, and prices on CM servers that are behind the firewall makes this information inaccessible and protects your data.

Scaling Sitecore E-Commerce Services

You can scale SES in two different ways:

- You can use the native Sitecore CMS 6.5 scalability features that allow you to run multiple Sitecore instances. This approach will primarily aim at running multiple CM instances behind a firewall and will allow you to run multiple CD servers on the other side of the firewall.
- You can use SES's service model to add multiple CD instances that manage all the web-requests as well as communicating with the backend CM servers that store the orders placed by customers and handle all the requests for price and stock information.

You can use either of these approaches on their own or combine them to facilitate more advanced setups.

Scaling SES with Sitecore CMS 6.5

Sitecore CMS 6.5 provides native support for configuring multiple related instances in order to achieve better load balancing on the CM servers. By using CMS 6.5 to scale SES, you can increase performance by having the content distributed across several CM servers.

For more information about using CMS 6.5 to configure SEFE, see the *Sitecore CMS 6.5 Scaling Guide*.

http://sdn.sitecore.net/Reference/Sitecore%206/Scaling%20Guide.aspx

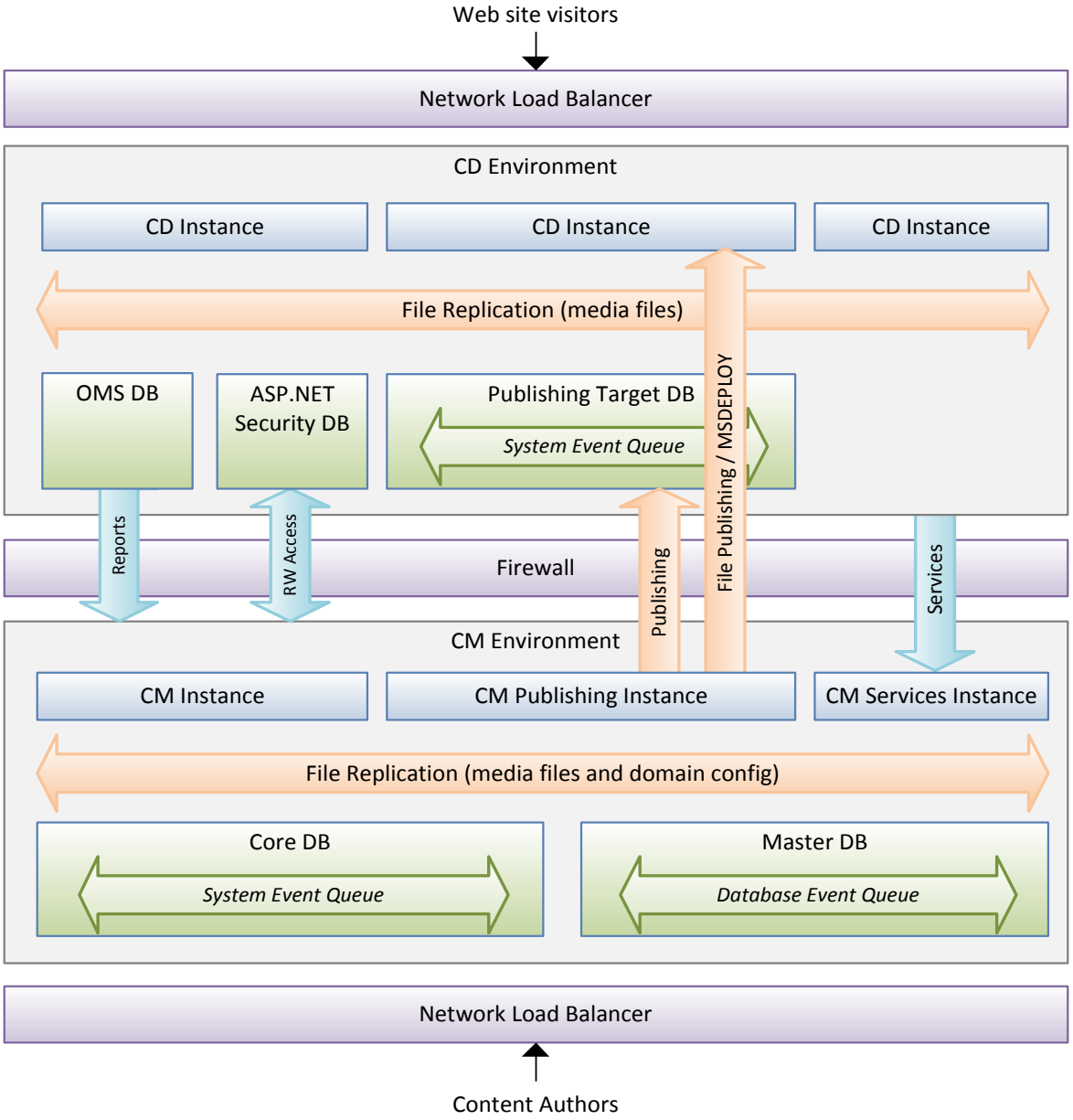
Scaling SES with the Service Model

SES contains an additional mechanism that you can use to scale your solution — the Service Model.

The Service Model allows you to avoid storing sensitive information such as orders, prices, and stocks on the CD servers. You can store this information on the CM servers that are protected by a firewall.

The Service Model consists of a number of service interfaces that are designed to link separate CM and CD servers. A service layer has been implemented using WCF technology that provides powerful configuration options in terms of communication protocols and security.

The following diagram shows the CD and CM instances and how the SES Service Model extends the CMS architecture.



The lower section illustrates the CM environment. The rightmost CM server is configured to host the services that the CD servers use to communicate across the firewall. The upper section illustrates the CD environment. The CD instances use the service layer to access the CM environment and manage orders, product stock, and price information.

The Separation of Content Management and Content Delivery Servers

The CD servers use the service layer to access the CM servers that contain the:

- Orders.
- Product price information.

- Stock information.

The service layer is divided into three separate services that corresponding with the previous list. You can use one or all of the services in this split CD and CM environment.

If you are running on a single server environment or CMS 6.5 scaled environment, none of the services are used. For more information, see also *SES Services Configuration* section.

Orders

You can store the orders that are placed by customers in Sitecore as it is done in the default implementation that comes with SES. Alternatively, you can store the orders in a 3rd party external system. If you store the orders in a 3rd party system, you must extend the part of the domain model that handles the orders.

Important
The CM servers should manage integration with the external systems. The CD servers should never do this. There is a single point of integration.

You must configure the *Order* service if you want to run separate CD and CM environments.

Product Price Information

The type of webshop you are running often determines where you should store the product price data.

B2C Shop

A business-to-consumer shop typically has fairly static prices. In which case, it might make sense to store the prices on the CD servers as well as on the CM servers. This is done by storing the product price data in the *Master* database and publishing the price in the same way as you publish ordinary CMS data. The *Example Pages* package demonstrates how you can store price data in the product template and publish it from there to the *Web* database.

B2B Shop

A business-to-business webshop typically has some more advanced price structures that can vary depending on a number of rules and the customer relationships that you support. The prices can be more variable in nature. It therefore makes sense not to store price data on the CD servers, but to request the price from the CM server through the service layer. In this case, the product price data should not be stored along with the product information as is done in the *Example Pages* package.

If the solution is integrated with a 3rd party system, the prices are typically stored there and the `ProductPriceManager` should be replaced with a customized version that integrates with the external system.

Important
The CM servers should manage integration with the external systems. The CD servers should never do this. There is a single point of integration.

Stock Information

Stock information fluctuates a lot and therefore it should never be stored on the CD servers. Stock information needs to be synchronized and the correct way to do that is on the CM server through the Service Layer.

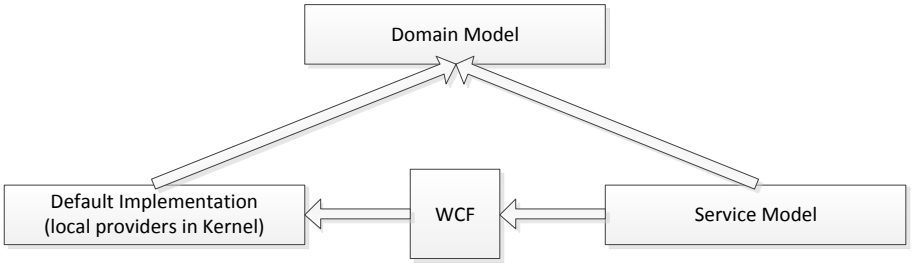
If the solution is integrated with a 3rd party system, stock information is typically stored there and the `ProductStockManager` should be replaced with a customized version that integrates with the external system.

Important
The CM servers should manage integration with the external systems. The CD servers should never do this. There is a single point of integration.

The Service Model

The SES domain model represents the E-Commerce abstraction layer and describes the API contracts.

Windows Communication Foundation (WCF) is used to implement the service model that is represented by a set of manager classes which implement the main Domain Model contracts.



The default implementation of the domain model is located in the `Sitecore.Ecommerce.Kernel` assembly and works with the local CMS instance and accesses the items that are local for that instance:

- OrderManager
- ProductPriceManager
- ProductStockManager

To setup the separated CD environment and use the service model, a set of additional remote managers have been added. These remote managers are located in the `Sitecore.Ecommerce.Services` assembly:

- RemoteOrderManager
- RemoteProductPriceManager
- RemoteProductStockManager

Both the *local* and *remote* managers have been setup in the `Unity.Config` file. By default the *local* managers are configured which means that they work with items from the local CMS instance.

To configure the remote managers, you should map the manager registration to the appropriate alias. In the `Unity.Config` file, locate the `<register>` elements section and change the `mapTo` attribute to the appropriate *remote* alias:

```
        <register type="IOrderManager" mapTo="RemoteOrderManager">
        ...
        <register type="IProductStockManager" mapTo="RemoteProductStockManager">
        ...
        <register type="IProductPriceManager" mapTo="RemoteProductPriceManager">
        ...
```

SES Services Configuration

Content Management (CM Server) Configuration

You must configure a CM server instance to host the SES Services. The CM server instance can share the *Master* and *Core* databases with other CM server instances in a CM environment.

You should configure the publishing targets of the CM server to point to the CD instances. For more information about configuring publishing targets, see the *Sitecore 6.5 Scaling Guide*.

Also in the `web.config` file of the CM instance, you must add the following section to the `<configuration>` section:

```
<system.runtime.serialization>
    <dataContractSerializer >
      <declaredTypes>
        <add type="Sitecore.Ecommerce.DomainModel.Products.ProductStock,
Sitecore.Ecommerce.DomainModel">
          <knownType type="Sitecore.Ecommerce.Products.ProductStock,
Sitecore.Ecommerce.Kernel"/>
        </add>
      </declaredTypes>
    </dataContractSerializer>
  </system.runtime.serialization>
```

Content Delivery (CD Server) Configuration

You must configure the Windows Communication Foundation (WCF) Service Model for every CD instance. You must configure each CD instance in the `web.config` file.

The CD configuration consists of a set of bindings, one for each service and a corresponding set of endpoints that map the bindings to a server instance. Also there must be registrations of known types that were shown in the previous section. In the following example, there are three bindings: `OrderService`, `ProductPriceService` and `ProductStockService`:

```
        <system.serviceModel>
        <bindings>
          <basicHttpBinding>
            <binding name="BasicHttpBinding_IOrderService" closeTimeout="00:02:00"
openTimeout="00:02:00" receiveTimeout="00:10:00" sendTimeout="00:02:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
              <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384" maxBytesPerRead="4096" maxNameTableCharCount="16384" />
              <security mode="None">
                <transport clientCredentialType="None" proxyCredentialType="None" realm=""
/>
                <message clientCredentialType="UserName" algorithmSuite="Default" />
              </security>
            </binding>
            <binding name="BasicHttpBinding_IProductPriceService" closeTimeout="00:02:00"
openTimeout="00:02:00" receiveTimeout="00:10:00" sendTimeout="00:02:00" allowCookies="false"
```

```
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
                <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384" maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                <security mode="None">
                  <transport clientCredentialType="None" proxyCredentialType="None" realm=""
/>
                  <message clientCredentialType="UserName" algorithmSuite="Default" />
                </security>
              </binding>
              <binding name="BasicHttpBinding_IProductStockService" closeTimeout="00:02:00"
openTimeout="00:02:00" receiveTimeout="00:10:00" sendTimeout="00:02:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
                <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384" maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                <security mode="None">
                  <transport clientCredentialType="None" proxyCredentialType="None" realm=""
/>
                  <message clientCredentialType="UserName" algorithmSuite="Default" />
                </security>
              </binding>
            </basicHttpBinding>
          </bindings>
          <client>
            <endpoint address="http://localhost/sitecore
modules/shell/ecommerce/services/OrderService.svc" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IOrderService" contract="OrderService.IOrderService"
name="BasicHttpBinding_IOrderService" />
            <endpoint address="http://localhost/sitecore
modules/shell/ecommerce/services/ProductPriceService.svc" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IProductPriceService"
contract="ProductPriceService.IProductPriceService"
name="BasicHttpBinding_IProductPriceService" />
            <endpoint address="http://localhost/sitecore
modules/shell/ecommerce/services/ProductStockService.svc" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IProductStockService"
contract="ProductStockService.IProductStockService"
name="BasicHttpBinding_IProductStockService" />
          </client>
        </system.serviceModel>
        <system.runtime.serialization>
          <dataContractSerializer >
            <declaredTypes>
              <add type="Sitecore.Ecommerce.DomainModel.Products.ProductStock,
Sitecore.Ecommerce.DomainModel">
                <knownType type="Sitecore.Ecommerce.Products.ProductStock,
Sitecore.Ecommerce.Kernel"/>
              </add>
            </declaredTypes>
          </dataContractSerializer>
        </system.runtime.serialization>
```

You can use the bindings as shown in this example but you must configure the endpoints to use the right URI (Uniform Resource Identifier). Endpoints give the clients access to the functionality of a WCF service that is hosted on a server instance. Each endpoint has an address attribute that is based on the URI protocol which has the following format:

`<scheme://hostname[:port]/path>`

Important
The http scheme is configured and each endpoint is given the `localhost` hostname by default. You must replace this with the address of the CM server that hosts the service model.

If the network configuration allows, you can change protocols used for binding the services to improve performance and security. You must configure this on both the CD and CM server instances. This is a feature of the WCF framework.

For more information about configuring the Service Model, see http://msdn.microsoft.com/en-us/library/ms731734.aspx.

For more information about configuring endpoints, see http://msdn.microsoft.com/en-us/library/ms733107.aspx.

Configuring the Lucene Search Index

SES uses Lucene for product indexing and searching by default. The search indexes configured in the `App_Config/Include/Sitecore.Ecommerce.config` file contain the configuration for both the *Master* and *Web* databases by default. In the CD environment, the servers are typically not associated with the *Master* database. Therefore, you should not configure the product index for the *Master* database on the CD server instances.

You must remove the highlighted section from the `Sitecore.Ecommerce.config` file:

`search/configuration/indexes/index[id="products"]/locations/master:`

```
<search>
  <configuration>
    <indexes>
      <index id="products" type="Sitecore.Search.Index, Sitecore.Kernel">
        <param desc="name">$(id)</param>
        <param desc="folder">__products</param>
        <Analyzer type="Sitecore.Ecommerce.Search.LuceneAnalyzer, Sitecore.Ecommerce.Kernel" />
        <locations hint="list:AddCrawler">
          <master type="Sitecore.Ecommerce.Search.DatabaseCrawler, Sitecore.Ecommerce.Kernel">
            <Database>master</Database>
            <Root>{0A702337-81CD-45B9-8A72-EC15D2BE1635}</Root>
            <Tags>master products</Tags>
          </master>
          <web type="Sitecore.Ecommerce.Search.DatabaseCrawler, Sitecore.Ecommerce.Kernel">
            <Database>web</Database>
            <Root>{0A702337-81CD-45B9-8A72-EC15D2BE1635}</Root>
            <Tags>web products</Tags>
          </web>
        </locations>
      </index>
    </indexes>
  </configuration>
</search>
```

SES Services Configuration Requirements

The service model does not require a specific CMS version and can therefore be used by any CMS version that SES supports.

WCF Configuration Notes

For more information about configuring WCF see:

http://msdn.microsoft.com/en-us/library/ms735093.aspx

http://msdn.microsoft.com/en-us/library/ms731884.aspx

http://msdn.microsoft.com/en-us/library/ms731316.aspx

http://www.devx.com/codemag/Article/33342/1763/page/1