Sitecore Experience Platform 8.2

Performance White Paper

sitecore® Own the experience™

Table of contents

Executive Summary

Sitecore Experience Platform (SXP) 8.2 performance testing focused on two main goals:

- Content delivery with a focus on Experience Analytics.
- The new publishing functionality.

Each performance testing sessions utilized a separate deployment with traffic modeled on realistic user interactions and key performance metrics monitored across the deployment. The following statements summarize the results of each test.

Content Delivery

- Testing ran for one hour with a user load that ramped from 110 to 800 users.

  User think time utilized a normal distribution profile based on 2 seconds.
- The testing did not request static files.

  In a typical enterprise deployment, static content is located in the content delivery network (CDN) or cached by a network appliance.
- The Sitecore Reference Storefront Powered by Commerce Server web site, using Microsoft ASP.NET MVC, was used as the test site.

  The deployment, therefore, included Commerce Server databases and related packages. The scenarios used in testing did not focus on Commerce functionality instead the load to the site drove the Experience Analytics functionality of Sitecore Experience Platform (SXP) 8.2.
- The *analytics* database on MongoDB server was configured using two shards to scale writing performance on the primary instances.
- Processing used two processing pool lanes to split the write capabilities of the *tracking_live* and *tracking_contact* databases across multiple disks.
- The database servers were configured with an *Extra Large A4* instance type.

  The SQL data and log files were stored on a hard drive that consisted of multiple virtual hard drives and the software was striped using Microsoft Storage Spaces.
- The MongoDB instances used Premium Disk to increase disk throughput.
- You can use the deployment techniques described in this document for on-premise and other Infrastructure as a Service (IaaS) cloud providers.
- The results are tightly linked to the software and virtual machine configurations used, including:
  - Instance size
  - Number of instances that make up each server farm
  - Amount of system memory on each instance
  - Database disk configuration
  - Virtual CPU capabilities

Publishing

- The new publishing engine increased publishing performance by up to 10 times for content items and 3.7 times for media items.
- The publishing manifest calculation process can take up to 3 times more resources than the other phases.
- The publishing process does not have any considerable impact on related Sitecore instances.

Content Delivery

Key Findings

- To achieve the best results, you should always follow the Azure SQL on VM and Sitecore CMS/DMS guidelines:
  - Performance best practices for SQL Server in Azure Virtual Machines

    https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/
  - Performance Guidance for SQL Server in Azure Virtual Machines

    http://msdn.microsoft.com/en-us/library/dn248436.aspx
  - CMS Performance Tuning Guide

    https://sdn.sitecore.net/Reference/Sitecore%207/CMS%20Performance%20Tuning%20Guide.aspx
- The size of each virtual machine has a significant effect on performance.
- The load on the content delivery web servers was distributed using Azure Load Balancer, which uses a round-robin algorithm to distribute the load.
- In a multi-region deployment, using the Azure Traffic Manager to ensure that the closest geographic location is recommended.

  For more information about Azure Traffic Manager, see:

  https://azure.microsoft.com/en-us/services/traffic-manager/
- Using Microsoft SQL Server page compression can improve disk throughput at the expense of CPU time.

  For these tests, page compression was configured for the databases that experienced significant write activity.
- Key test results

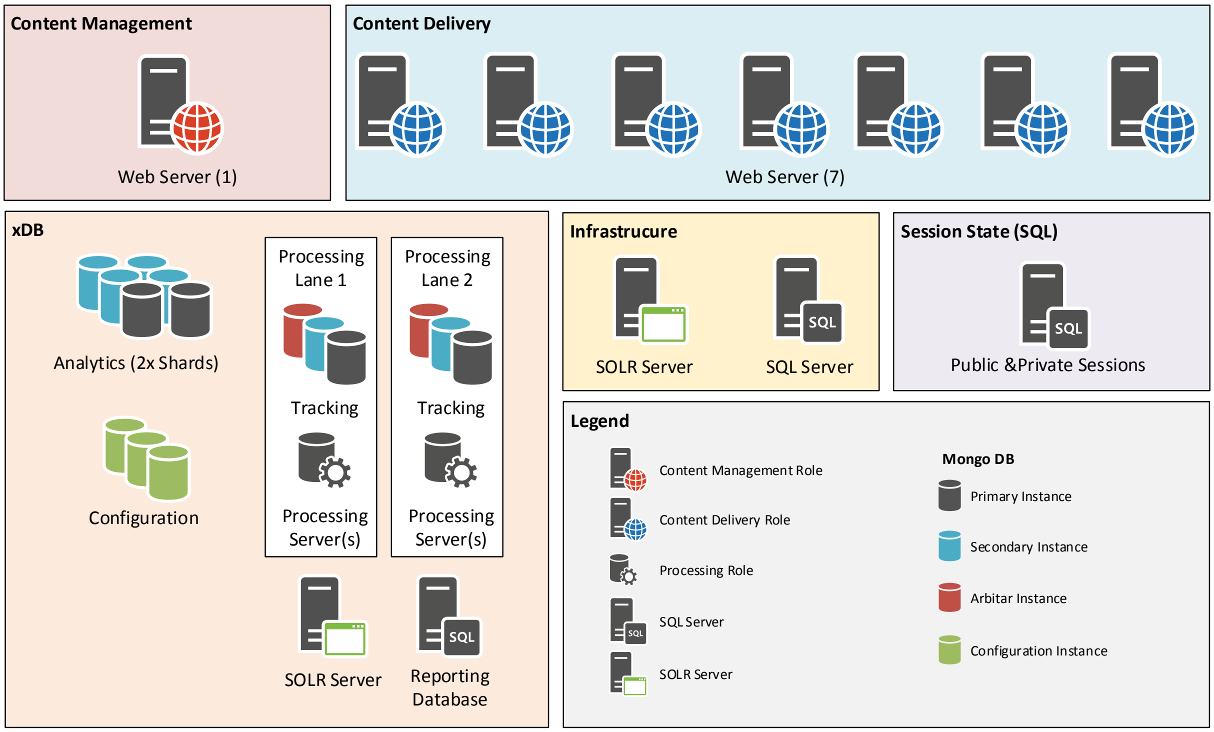
  The following results summarize the one-hour test run:
  - Total Page Views: 991,812
  - Total Requests: 1,987,531
  - Total Visits: 34,232
  - Average response time: 0.37 seconds
  - Max number of simultaneous user load: 800 users
  - Average % CPU processor at max load: 85.79%

Deployment

Testing utilized a Sitecore Experience Platform scaled configuration with all the server roles on separate instances to maximize scalability.

In this document, we refer to the Internet-facing load balanced web servers as Content Delivery(CD) servers. The Azure Load Balancer was used to balance the load to all of the individual CD instances. This load balancer used a round-robin algorithm to distribute traffic.

The Content Management (CM) portion of the deployment was a single server and did not use a load balancer. Sitecore does allow the CM to scale horizontally.



For more information about the supported Sitecore CMS deployments, see the Sitecore CMS Scaling Guide:

http://sdn.sitecore.net/Reference/Sitecore%207/Scaling%20Guide.aspx.

Virtual Machine Instance Size

The following settings were applied to the Microsoft Azure Virtual Machine instances:

Virtual machine	Instance size
Web Server Instance	Extra large (A4) instance • Includes: Content Management, Content Delivery, and Processing instances • 14 GB memory • 8 virtual cores • 64-bit platform
Database Instance	Extra large (A4) instance • 14 GB memory • 8 virtual cores • Database disk (containing data and log files); sixteen 25GB virtual hard disks, software striped with 64-KB cluster size using Storage Spaces • 64-bit platform
MongoDB Instance	Large (DS3) instance • 7 GB memory • 4 virtual cores • 64-bit platform • Configured with Premium Disk Storage Accounts (SSD-based arrays) to maximize write performance
Search and Indexing Instance	Large (A3) instance • 7 GB memory • 4 virtual cores • Index disk is made up of four 100GB disks striped together with a cluster size of 64-KB using Storage Spaces • 64-bit platform
Load Test Instance	Large (A3) instance • 1 controller • 5 agents Followed Microsoft's guidance: http://msdn.microsoft.com/en-us/library/ff937706.aspx

Azure Configuration

The following settings applied to Microsoft Azure:

Azure Configuration	Settings
SQL Server	**SQL Server in Virtual Machine Performance Guidance** Both of the SQL servers in this deployment were configured for the Online Transaction Processing (OLTP) workloads. • https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/ • http://msdn.microsoft.com/en-us/library/dn248436.aspx **Tables with SQL page compression enabled** The following tables that are write-heavy had SQL Page Compression enabled to optimize disk usage: • Sitecore Analytics. • Sitecore Commerce Transaction Table. **Note** Compression has an impact on CPU usage. **Disk configuration** • Combine the disk with storage spaces using a 64-KB cluster size. For more information about Microsoft Storage Spaces, see: http://social.technet.microsoft.com/wiki/contents/articles/15198.storage-spaces-overview.aspx
MongoDB Server	• Premium Storage was used to optimize the disk's write capabilities. For more information about premium disks, see: https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/
Storage Accounts	The storage accounts that hosted the virtual hard disks and data disks were split as follows: • Infrastructure: Content Management, Directory Server • Main SQL database: Sitecore and Sitecore Commerce Databases • Reporting SQL database: Reporting databases • Content Delivery: Content Delivery • Index: SOLR instances • Mongo (Premium Storage Account): MongoDB instances

Software Configuration

The following software was used:

Server	Software

Web Server
• Content Management
• Content Delivery
• Processing
	• Microsoft Windows Server 2012 R2 Datacenter
	• Microsoft Internet Information Services (IIS) 8.5 (Integrated Mode/.NET Framework v4.0)
	• Sitecore CMS 8.2
	   o Sitecore.Solr.Support
	   o Sitecore Commerce Connect 8.2
	• Sitecore Commerce 8.2
	   o Sitecore Commerce Server Connect 8.2
	   o Sitecore Reference Storefront Powered by Commerce Server 8.2

Database Server (SQL)	• Microsoft Windows Server 2012 R2 Datacenter • Microsoft SQL Server 2014 Enterprise
Database Server (Mongo)	• MongoDB 3.2.4
Search and Indexing Server	• Microsoft Windows Server 2012 R2 • Apache SOLR version 5.1.0 • Java Runtime Environment (JRE) was 1.7 (64-bit edition)
Load Test Servers	• Microsoft Visual Studio 2013 Ultimate (Update 4) • Microsoft Visual Studio 2013 Test Controller (Update 4) • Microsoft Visual Studio 2013 Test Agent (Update 4)

Role Configuration

The following settings applied to Sitecore and its components.

Configuration	Settings
Content Delivery	**CMS Performance Tuning Guide** https://sdn.sitecore.net/Reference/Sitecore%207/CMS%20Performance%20Tuning%20Guide.aspx **Settings** • Session Timeout `sessionState[timeout]` setting configured in the `web.config` was set to the minimal value of `1`. **Note** This value is a test specific value that forces user sessions to timeout quickly. Sessions timing out allows the load to be applied to the processing infrastructure. This value is a business decision based on how long you think that sessions should exist. **Recommended Settings** • The `Render Layout Cache` setting was configured to cache where applicable for the Sitecore Reference Storefront Powered by Commerce Server website. • In the `sitecore.config` file, the value of the `Caching.DisableCacheSizeLimits` was set to `false`. You typically use this setting to tune cache sizes to acceptable levels during development. It is best practice to set this setting to `false` in production environments unless a Sitecore Support representative has instructed you to set it to `true` to address a specific problem. • Sitecore log file writes are kept to a reasonable level. In the `sitecore.config` file, in the `logger` section, the `level value` was set to `WARN`.

Configuration	Settings

Analytics Database	**xDB Configuration Guide**
	https://sdn.sitecore.net/SDN5/Reference/Sitecore%207/xDB%20Configuration%20Guide.aspx
	• The *analytics* MongoDB database can be sharded to increase its scaling capabilities. For this test, the MongoDB instance consisted of two shards.
	• A combination of MongoDB instances and processing servers collectively called Processing Lanes, can be deployed to increase the scalability of the processing tier.
	For these tests, two processing lanes that made up of a single processing role were hosted in an A4 instance.
	• The processing lanes were split, with 4 CD servers configured to use one lane and the remaining 3 CD servers connected to the other lane.
	  ○ The 1st processing lane's databases were hosted on a P20 Premium disk.
	  ○ The 2nd processing lane's databases were hosted on a P10 Premium disk.
	
	For more information about Azure Premium Disks, see:
	https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/

Processing	• The tested deployment's processing capacity was split into two lanes to increase processing scale. This configuration spreads the disk writes across multiple instances of MongoDB. 4 content delivery servers wrote to one MongoDB instance, and the remaining 3 wrote to the second instance. **Note** It is also possible to scale processing servers horizontally by configuring more than one processing instances against a single MongoDB instance. This type configuration is useful when CPU time is the limiting resource.

Methodology

Site Specification

Configuration	Values
Web technology	ASP.NET MVC
Number of pages	Over 1 million (product detail pages)
Multi-language	3 languages
HTML caches are used	True
Number of page templates	27
Number of user profiles	10 million

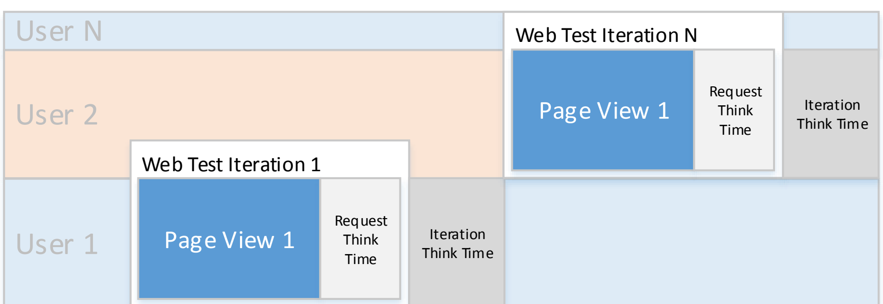
Test Scenarios

Performance testing employed the following scenarios:

Anonymous with a single interaction and a single page view per interaction

This scenario simulates anonymous visits to the website that create a single interaction with a single page view.

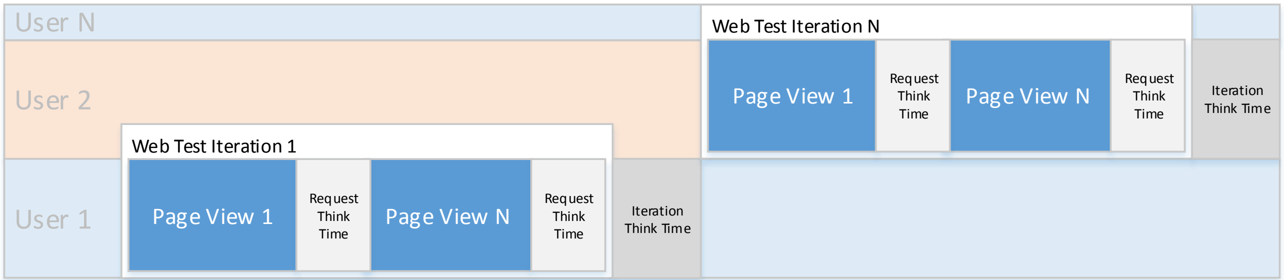
1. Browse to a single page.
2. Clear your cookies and browse to a new page.



Anonymous with a single interaction and many page views per interaction

This scenario simulates anonymous visits to the website that create a single interaction with multiple page views.

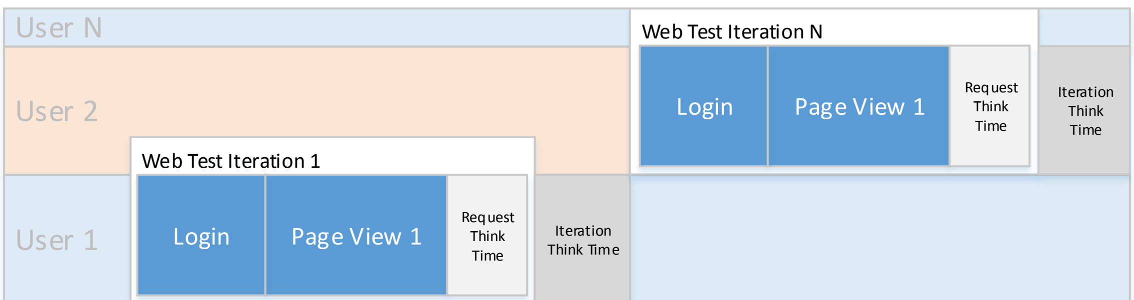
1. Browse to 10 random pages.
2. Clear your cookies and browse to 10 new pages.



Authenticated with a single interaction and a single page view per interaction

This scenario simulates authenticated, or identified, visits to the website that create a single interaction with multiple page views.

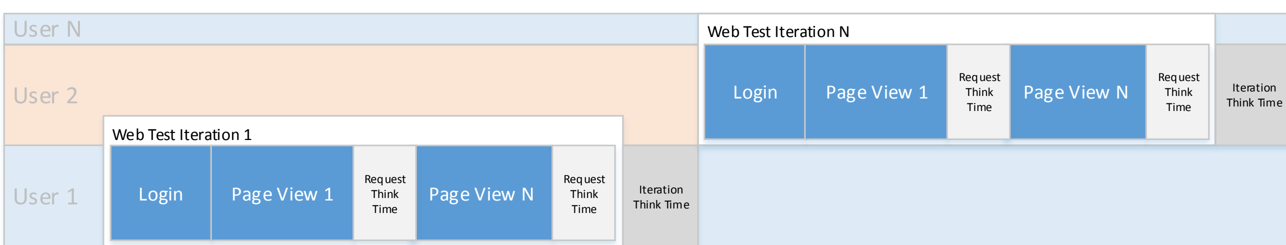
1. Trigger the login page, which calls Analytics Identification using Tracker.Current.Session.Identify()
2. Browse to a single page.
3. Clear your cookies and browse to a new page.



Authenticated with a single interaction and many page views per interaction

This testing scenario simulates authenticated, or identified, visits to the website that create a single interaction with multiple page views.

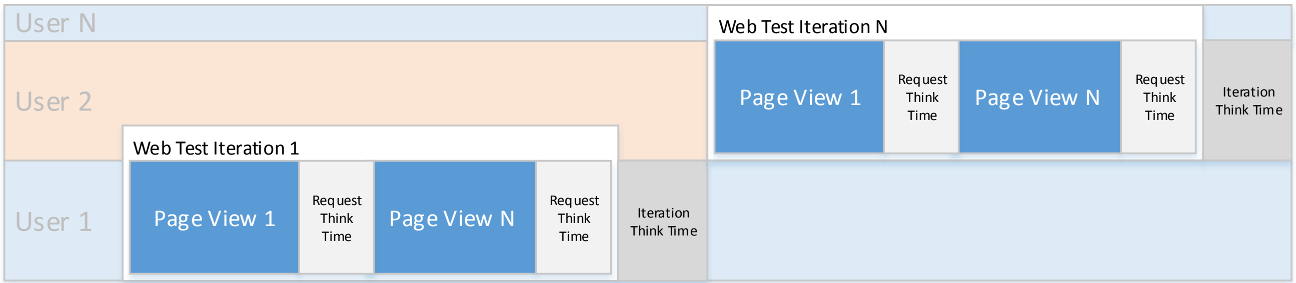
1. Trigger the login page, which calls Analytics Identification using Tracker.Current.Session.Identify()
2. Browse to 30 random pages.
3. Clear your cookies and browse to multiple pages again.



Anonymous with a single interaction and a significant number of page views per interaction

This scenario simulates anonymous visits to the website that create a single interaction with a large number of page views. This testing scenario simulates a robot interacting with the site.

1. Browse to 200 random pages.
2. Clear your cookies and start browsing again.



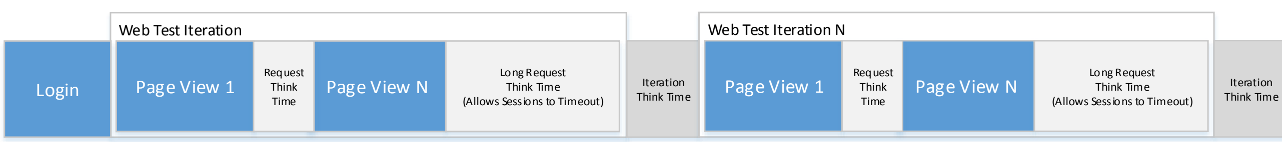
Anonymous with multiple interactions and multiple page views per interaction

1. Browse to 10 random pages.
2. Wait 240 seconds to allow the session to timeout and then repeat.

Authenticated with multiple interactions and multiple page views per interaction

At the start of each user thread, trigger the login page, which calls Analytics Identification using `Tracker.Current.Session.Identify()`

1. Browse to 10 random pages.
2. Wait 240 seconds to allow the session to timeout and then repeat the process.



Load Matrix

Load configuration

The following table lists the load that was used in each scenario:

Scenario group	Interactions	Page views per interaction	Percentage of traffic	Number of users	Authentication type
Single interactions	1	1	20%	160	Anonymous
	1	10	50%	400	Anonymous
	1	200	10%	80	Anonymous
	1	1	5%	40	Authenticated
	1	10	5%	40	Authenticated

Scenario group	Interactions	Page views per interaction	Percentage of traffic	Number of users	Authentication type
Anonymous multiple interactions	Multiple	10	5%	40	Anonymous
Authenticated multiple interactions	Multiple	10	5%	40	Authenticated

User specifications

Configuration	Values
Max number of users	800
Think time	2 seconds
Single interaction group	Step • Initial user count: 100 • Maximum user count: 720 • Step duration in seconds: 150 • Stamp ramp time in seconds: 30 • Step user count: 100 • Percentage new user: 0 • Think profile: normal distribution
Anonymous multiple interactions group	Step • Initial user count: 5 • Maximum user count: 40 • Step duration in seconds: 150 • Stamp ramp time in seconds: 30 • Step user count: 5 • Percentage new user: 100 • Think profile: normal distribution
Authenticated multiple interactions group	Step • Initial user count: 5 • Maximum user count: 40 • Step duration in seconds: 150 • Stamp ramp time in seconds: 30 • Step user count: 5 • Percentage new user: 100 • Think profile: normal distribution

Run settings

- Duration: 1 hour
- Warm up: 5 minutes
- Web test connection model: Connection per user
- Dependent request: Off

Data Configuration

Catalog data (Sitecore Commerce items)

- Products: 1 Million – 50% products, 50% product families
  - Product families contained a random number of variants from 1-20.
  - 50 % of all products and product families included between 1-4 relationships.
- Categories: 11,000
- Category Depth: 5
- Products/Category: 100

User Profiles

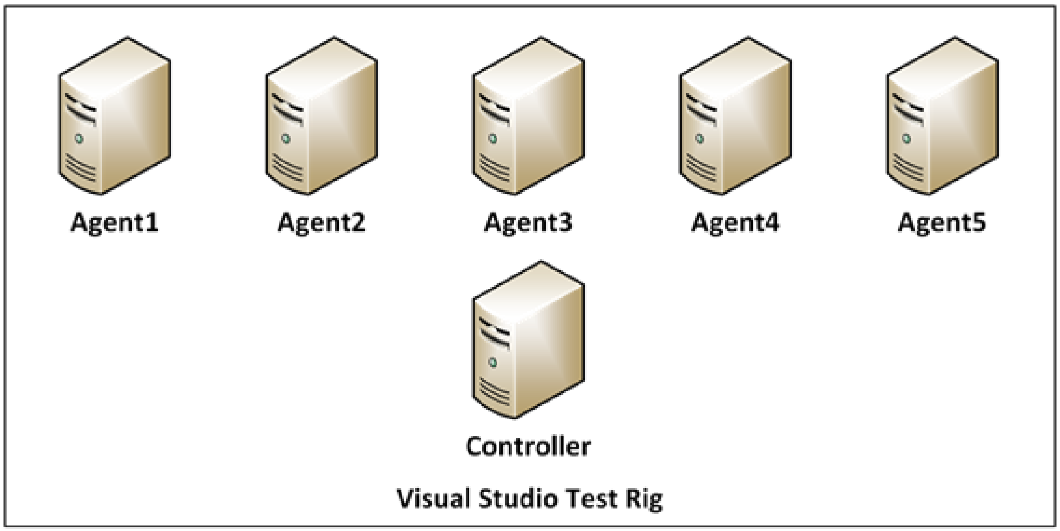
- 10 million users.
  - Each commerce user includes one shipping address and four credit cards.
  - Each credit card includes one billing address
- 40 million credit cards
- 50 million addresses
- A total of 100 million profiles objects were used.
- All the commerce profiles were imported into Sitecore.

Test Infrastructure Configuration

To generate the performance load, this test used a Visual Studio 2013 Load Test Rig with the following configuration:

- The Visual Studio 2013 test agents and the controller are deployed to Azure Large A3 instances.
- Each agent had a single data disk attached, and the agent service was set to utilize the attached disk as the working directory ensuring that the agent had enough disk capacity to handle load generation.

- For more information on setting up the Visual Studio Test agents and controllers, visit: http://msdn.microsoft.com/en-us/library/dd648127.aspx



Performance Metrics

Name	Performance counter	Description
Disk Idle Time	LogicalDisk(*)\% Idle Time	The percentage of time that the disk system was not processing requests and no work was queued.
Processor Time	Processor Information(_Total)\% Processor Time	The primary indicator of the processor activity.
Request Queued	ASP.NET\Requests Queued	The number of requests that are currently in the queue.
Page Time	-	The average time a page responds to the test agent.
Batch Requests	SQLServer:SQL Statistics\Batch Requests/sec	The number of Transact-SQL batch requests received by the server per second. This statistic is affected by all constraints such as I/O, the number of users, cache size, complexity of requests, and so on. High values mean good throughput.
Network Throughput	Network Interface\Bytes Total/sec	Shows the rate at which bytes are sent and received on the network interface, including framing characters. Bytes total/sec is the sum of the values of Network Interface\Bytes Received/sec and Network Interface\ Bytes Sent/sec.
Page Throughput	-	The number of pages per second, measured by the test agent

Name	Performance counter	Description
Process Memory	Process\Private Bytes	The current byte size of the virtual address space that the process is using.
Request Throughput	-	The number of requests per second measured at the test agent. This number includes all the requests to the test deployment; page requests and any dependent requests that were transmitted are included.
User load	-	The current number of users.
Virtual Memory	Process\Virtual Bytes	The current size, in bytes, of the virtual address space that the process is using.
Working Set	Process\Working Set	The set of memory pages or areas of memory allocated to a process that was recently used by the threads in the process.

Results

General Performance Findings

- For the tested scenarios, the main performance bottleneck on the CD servers was processor time.

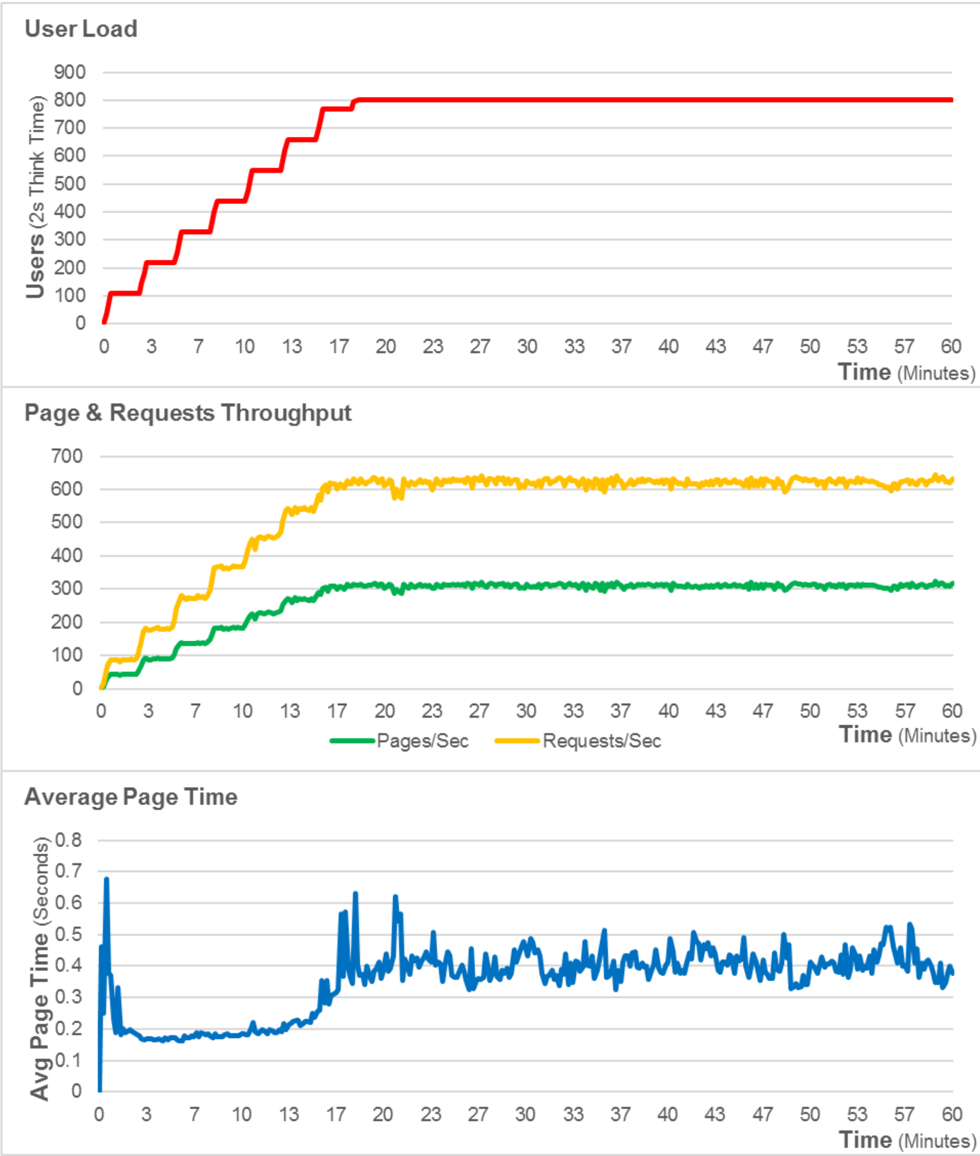
  Increasing the number of cores, core speed, and the number of server instances will allow the site to scale further, provided there is enough capacity on the MongoDB and SQL tiers.

- The out of process session state plays a major role in the performance capacity of the CD servers. Using the Session Database Performance Script, which places sessions in the temp database, greatly improves performance and scale.
- Azure Premium disk was used to scale the Analytics and Processing Servers. On-premise deployments should consider using SAN or SSD-based disks for similar throughput results.
- The session databases were hosted on Premium Disk. The shared and private sessions were located on a different disk because Azure disks do not perform well with sequential disk writes such as database log files.
- MongoDB primaries need fast disks to perform and scale well. Testing determined that having at least one secondary with a similar disk configuration ensures that data is safely synced in a timely fashion.
- The MongoDB processing instances can act as a bottleneck for a deployment’s CD farm. When this is the case, and depending on the Processing Role configuration, the primary processing disk can be overwhelmed to the point that front-end servers slow down, due to the time it takes for updates to be made to the MongoDB database. The time to make updates is directly dependent on the underlying disk capabilities.

  CD server writes are related to the number of private and shared sessions expiring. One way to scale the MongoDB processing capabilities is to add multiple processing lanes, which effectively spreads the writes across multiple MongoDB instances.

- MongoDB instance throughput can be improved by ensuring that databases and journaling are on different disks.

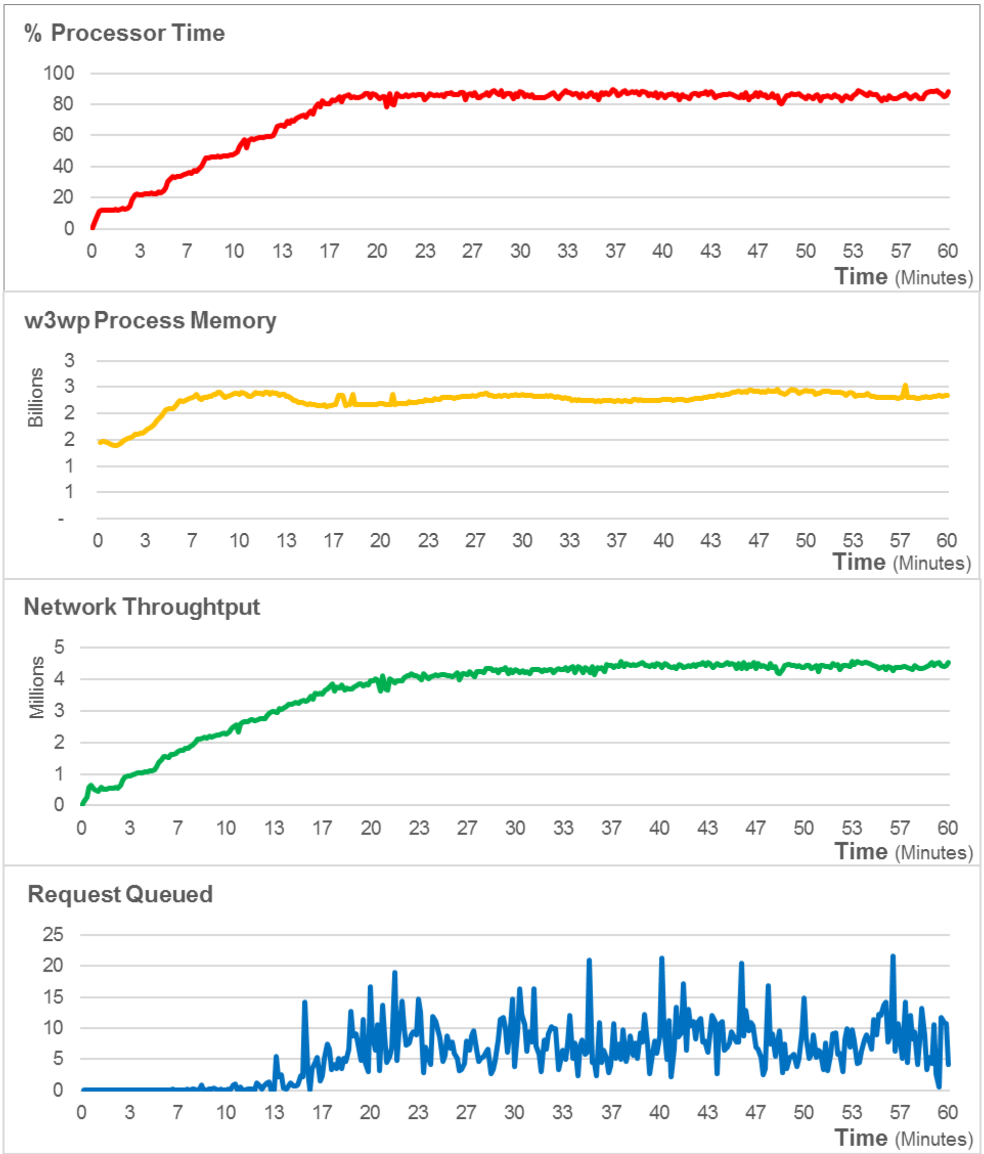
Key Metrics



These charts display the peak number of pages per second and the average page time. Most of the pages requested during testing consisted of 2 requests to the server, the page request, and the visitor identification request.

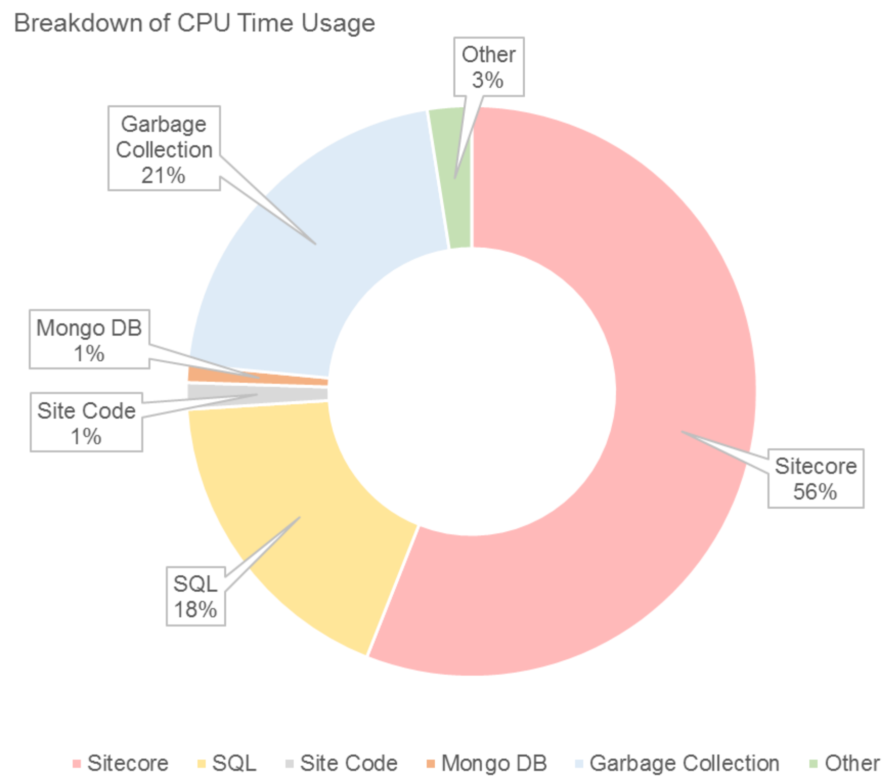
Testing resulted in a maximum page per second value of 323 pages, based on 645 requests, with a maximum page response time of 0.68 seconds and an average of 0.37 seconds. The worst performing page was the Login Submit page, with an average page time of 1.23s.

Delivery: Web Server



At maximum load, the average processor time was 85.8%, the average Network Bytes/Second was 4.3 MB/Second, and the number of ASP.NET Request Queued was 8. Because the Processing Queue's MongoDB instance played a role in the capacity of the CD servers, a second processing lane was created in the test deployment.

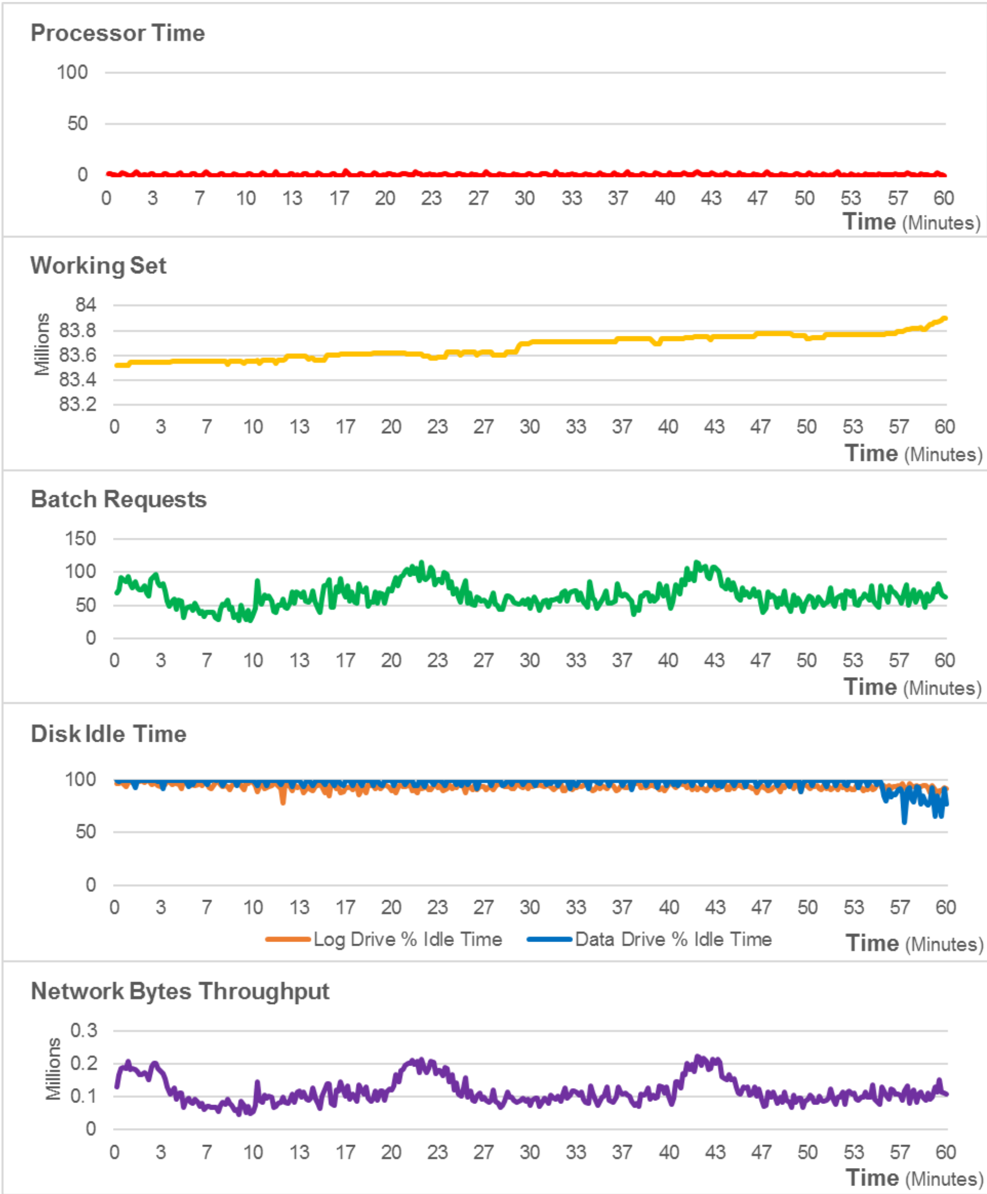
Delivery: Web Server CPU Usage



Note:

- This chart is created using a CPU trace from a single content delivery server in sampling mode.

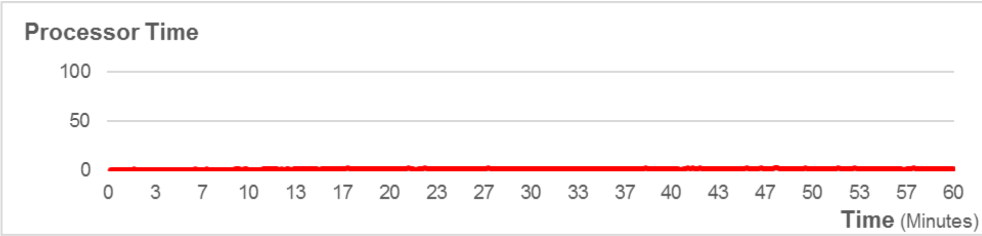
Site: SQL Server (Web, Master, Core)

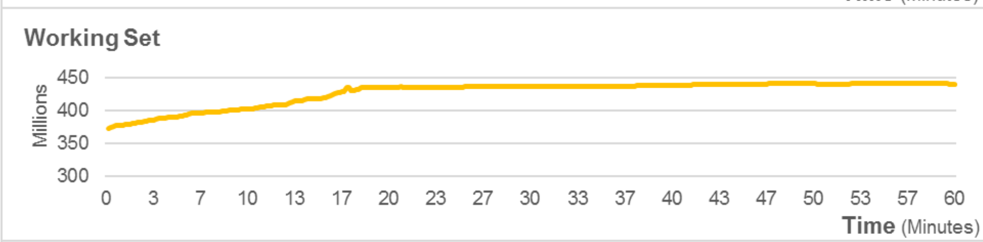


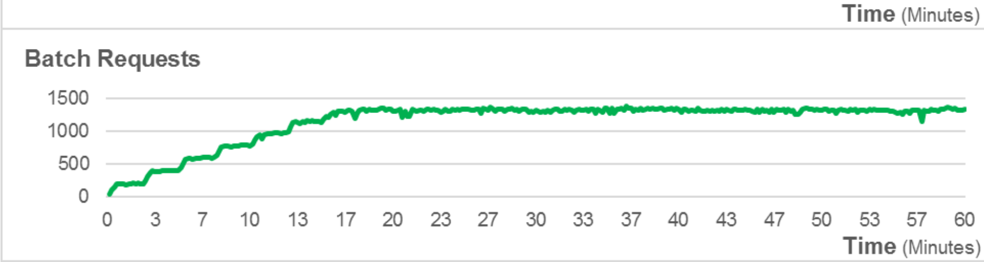
The site database server was more than capable of handling the load generated by the CD servers in the test run.

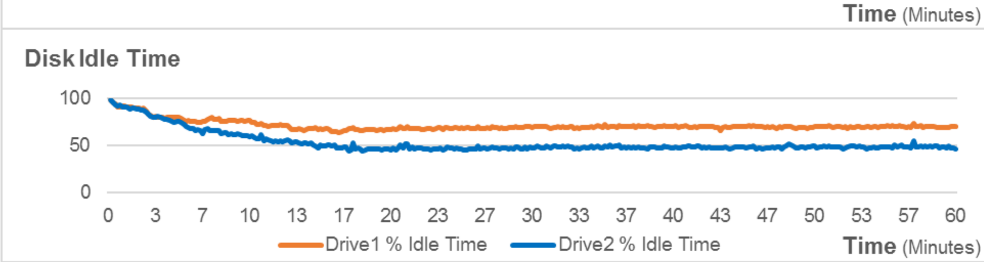
Note
The Sitecore Reference Storefront Powered by Commerce Server was configured with the optimal HTML cache settings. This configuration greatly impacts the database server load.

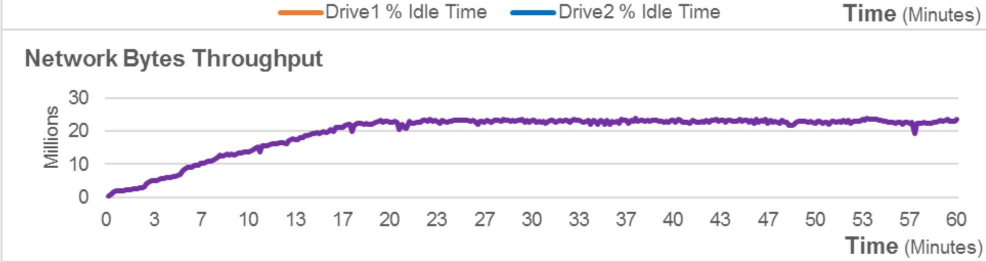
Session: SQL Server





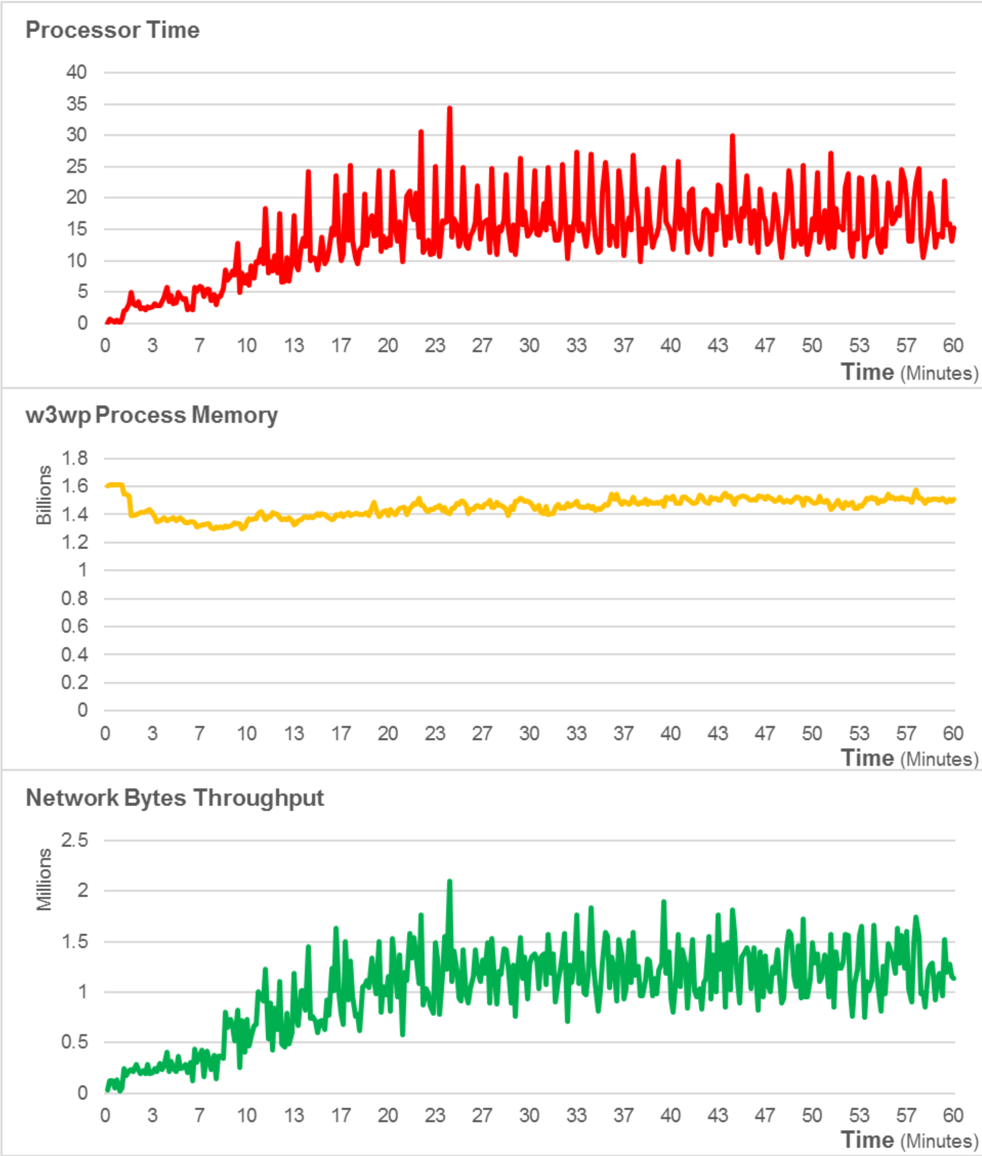






The session databases were split onto different disks for private and shared sessions. The session databases used synonyms to point to the data in the temp tables, which greatly optimized the session data usage. The testing produced a peak of 1,374 batch requests a second during the test run. To configure the process of using synonyms in the session databases, run the following script: Session Database Performance Script.

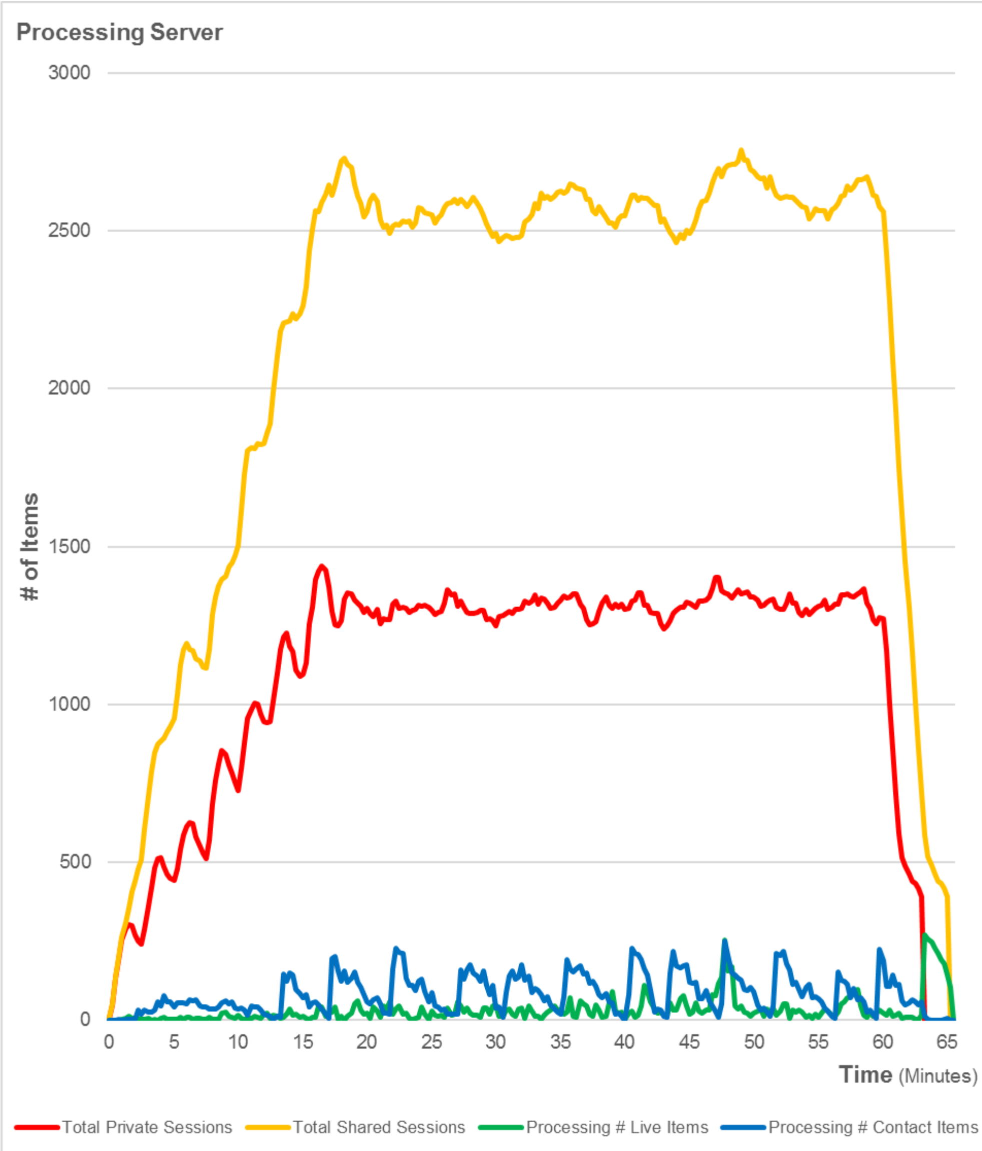
Processing: Web Server



These charts are an average of the 2 processing servers for this test deployment. Processing server usage depends on the rate at which contacts and interactions are created, which depends directly on the session timeout setting defined in the `web.config`. In this test run, multiple users performed multiple page views, and this did not represent the worst case scenario regarding creating raw interactions.

In additional testing, which is not described in this whitepaper, it was determined that a load scenario with all new users performing a few page views could have significantly more impact on processing infrastructure. When determining the size of the processing tier, if the expected load profile includes many users with a low number of page views per user, you should consider having more processing lanes or processing servers to ensure that items are processed efficiently during peak usage.

Processing: Sessions and Processing Pools

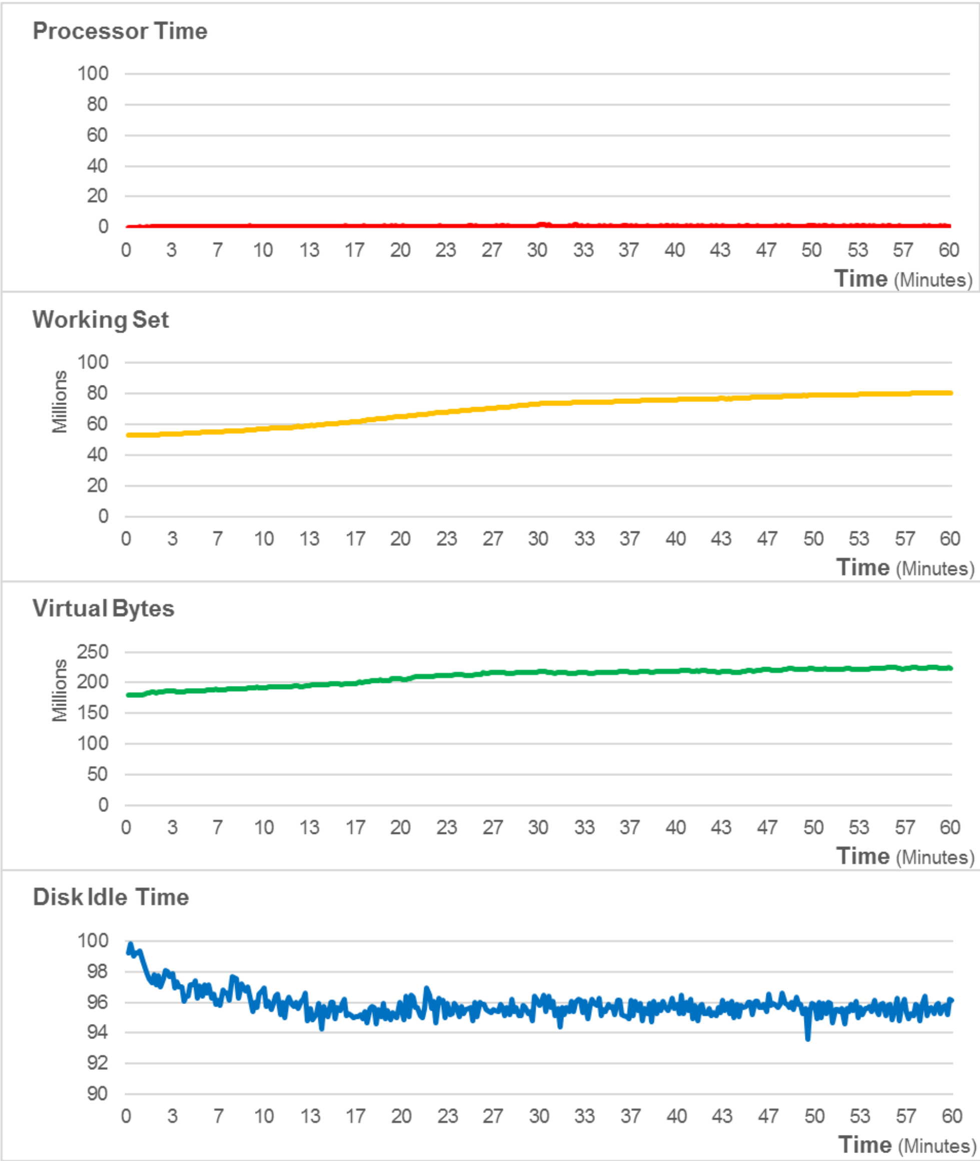


This chart displays the relationship between the session (SQL) and the tracking queues (MongoDB). This chart only includes the larger processing lane. The session count is distributed across the seven CD servers. The tracking count includes session expirations from four of these CD servers. Processing is highly dependent on the capacity of the SQL server that is hosting the reporting database; this is why a separate instance is used in this deployment.

The Aggregation setting is defined in the `Sitecore.Analytics.Processing.Aggregation.Services.config` file. The following were active in this run:

- aggregator: MaxThreads set to16
- contactProcessing: MaxThreads set to 4

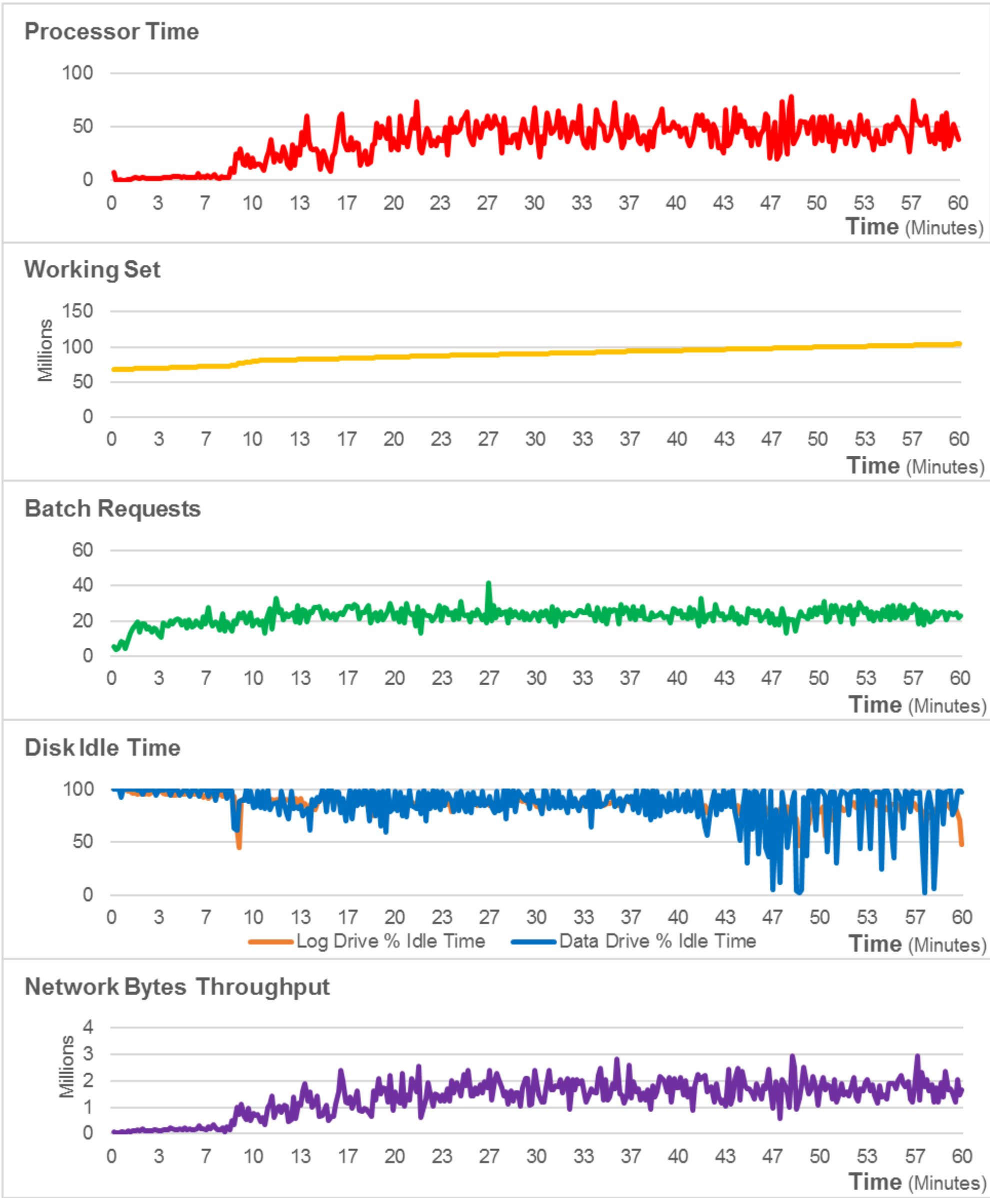
Processing: MongoDB Server



In our test deployment, we used two processing lanes. One lane handled four CD servers while the other lane handled the remaining three. In this configuration, both lanes utilized an Azure Premium Disk with a size of P10. These results are averaged across the two server instances.

The average % processing time was not an issue during this test run. Under an extreme load, we found that the CPU usage can spike, which typically indicated that disk subsystem was not able to keep up with the incoming load. Possible solutions for this issue include separating the journal and the data files across separate disks, using the Wired Tiger storage engine, and increasing the number of processing lanes to split the disk writes.

Reporting: SQL Server

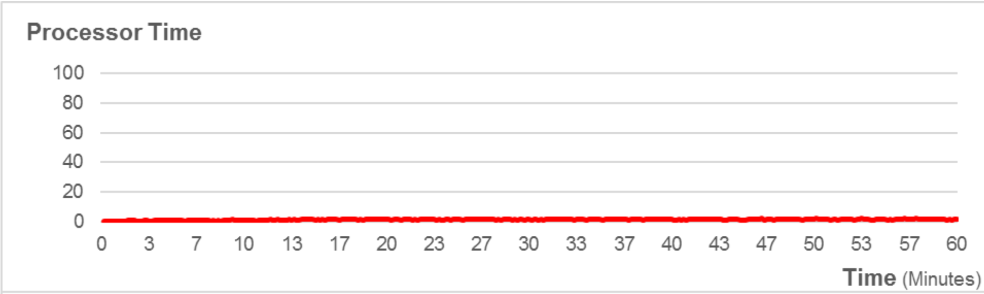


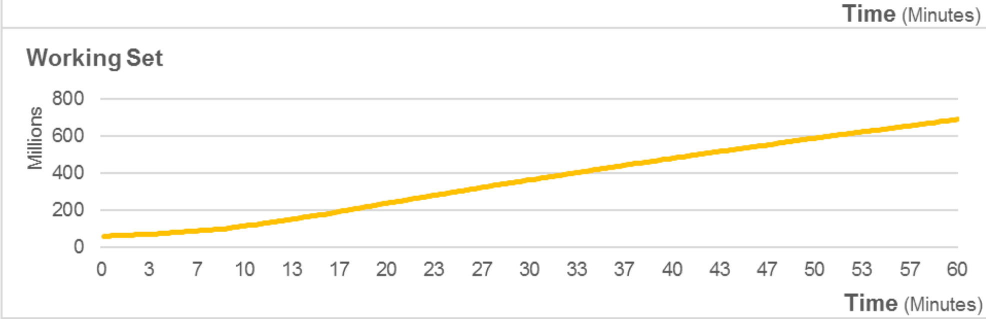
Our testing found that the reporting server can limit the total processing throughput capability.

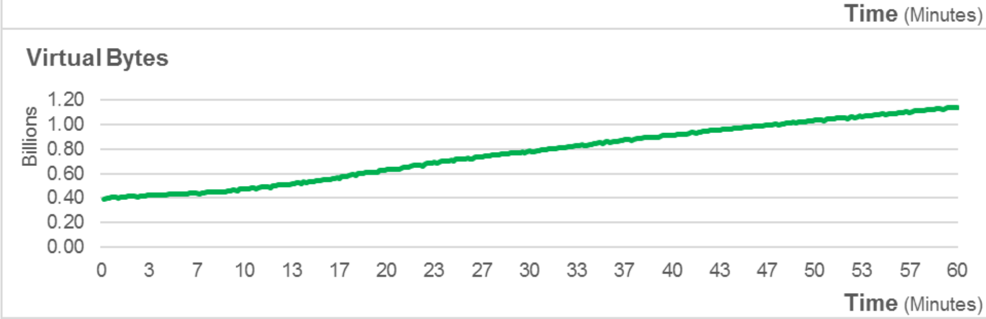
The SQL server configuration followed this guide:

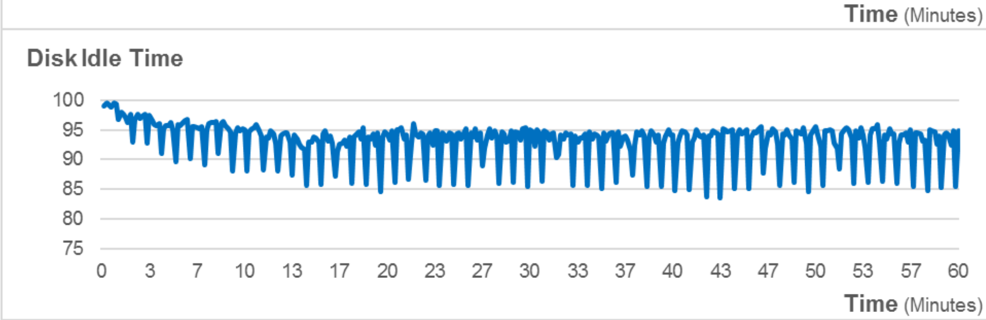
https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/

Analytics: MongoDB Server









The primary instances used instances of a P30 Azure premium disks.

In this test, CPU usage was not an issue. In testing done outside of this Whitepaper, we found the CPU can start to spike, usually indicating that the disk subsystem was not keeping up with the incoming load. If this becomes an issue, the you can take following actions to increase throughput:

- Increasing the number of shards
- Place the journaling and data on separate disks.

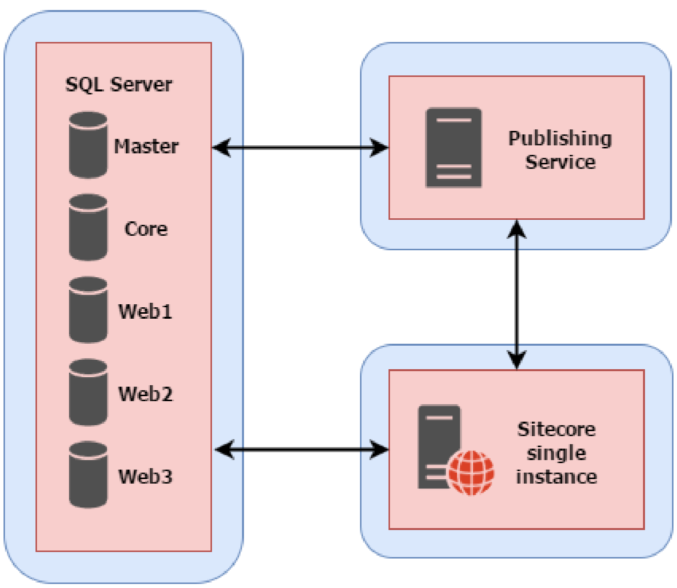
Publishing

Key Findings

- Publish operations were run using the Publish Item (Include subitems) option.
  - Publishing content items was found to be around 10 times faster when compared to traditional publishing.
  - Publishing media items was found to be around 3.7 times faster when compared to traditional publishing.
- Publishing process does not have a considerable impact on the Sitecore instance
- Average % CPU Processor for Sitecore Publishing Service at Max Load: 75%
- Average RAM usage for Sitecore Publishing Service at Max Load: 500MB

Deployment

- Sitecore Experience Platform single configuration was used in the performance testing of the Sitecore Publishing Service.
- In this configuration, SQL Server, the Publishing Service, and a Sitecore single instance are installed on separate machines.
- The testing focused on the performance of item publishing using the Publishing Service, measuring the time it took and comparing it with the standard Sitecore publishing functionality.



Virtual Machine Instance Size

This following settings were applied to Microsoft Hyper-V hosted virtual machine instances:

Virtual machine	Instance size
Web server instance	• Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz • 11.7 GB memory • 4 virtual cores • Single virtual hard disk
Database instance	• Intel(R) Core(TM) i7-4771 CPU @ 3.50GHz • 12.2 GB memory • 6 virtual cores • Single virtual hard disk
Publishing instance	• Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz • 12.2 GB memory • 6 virtual cores • Single virtual hard disk

Software Configuration

Server	Settings
Web server	• Microsoft Windows Server 2012 R2 Datacenter • Microsoft Internet Information Services (IIS) 8.5 (Integrated Mode/.NET Framework v4.0) • Sitecore CMS 8.2
SQL server	• Microsoft Windows Server 2012 R2 Datacenter • Microsoft SQL Server 2014 Enterprise
Publishing server	• Microsoft Windows Server 2012 R2 Datacenter • Microsoft Internet Information Services (IIS) 8.5 (Integrated Mode/.NET Framework v4.0) • .NET Core v1.0

Sitecore Configuration

Configuration	Settings
Sitecore Experience Platform 8.2	https://dev.sitecore.net/Downloads/Sitecore_Experience_Platform/82/Sitecore_Experience_Platform_82_Initial_Release.aspx
Sitecore Publishing Service	https://dev.sitecore.net/Downloads/Sitecore_Publishing_Service/11/Sitecore_Publishing_Service_11_Initial_Release.aspx

Role Configuration

Configuration	Settings
Database	General SQL Tuning: https://msdn.microsoft.com/en-us/library/ff647793.aspx

Methodology

Test Scenarios

The performance testing employed the following scenario groups:

- Publishing Content Items: Publishing a large amount of content items.
- Publishing Media Items: Publishing a large amount of media items with different types of media files.

Data Configuration

Scenario Group	Sitecore Items	Targets
Content	• 125,000 content items i) 5 shared fields ii) 5 versioned fields iii) 5 un-versioned fields iv) 3 languages v) 3 versions per language	3 Web databases
Media	• 27,000 media items i) Every item has a media file ii) 3 languages iii) 1 version for every language	3 Web databases

Performance Metrics

Name	Performance counter	Description
Disk Idle Time	LogicalDisk(*)\% Idle Time	The percentage of time that the disk system was not processing requests and no work was queued.
Processor Time	Processor Information(_Total)\% Processor Time	The primary indicator of the processor activity.
Process Processor Time	Process\% Processor Time	

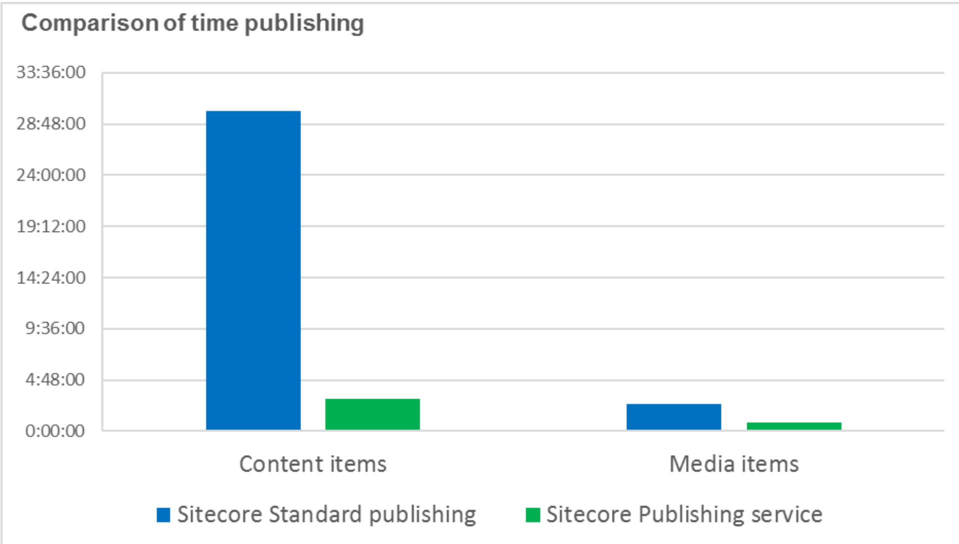
Name	Performance counter	Description
Batch Requests	SQLServer:SQL Statistics\Batch Requests/sec	The number of Transact-SQL batch requests received by the server per second. This statistic is affected by all constraints such as I/O, the number of users, cache size, complexity of requests, and so on. High values mean good throughput.
Network Throughput	Network Interface\Bytes Total/sec	Shows the rate at which bytes are sent and received on the network interface, including framing characters. Bytes total/sec is the sum of the values of Network Interface\Bytes Received/sec and Network Interface\ Bytes Sent/sec.
Process Memory	Process\Private Bytes	The current byte size of the virtual address space that the process is using.
Disk Activity	PhysicalDisk(_Total)\Avg. Disk sec/Write PhysicalDisk(_Total)\Avg. Disk sec/Read	Displays the average time the disk transfers took to complete, in seconds. Although the scale is seconds, the counter has millisecond precision, meaning a value of 0.004 indicates the average time for disk transfers to complete was 4 milliseconds. This counter is used to measure IO latency.

Results

General Performance Findings

- At the beginning of the publishing process, during manifest calculation, the Publishing Service may take about 3 times more resources than during the remaining phases of publishing.
- The publishing process does not have much of an impact on the Sitecore instance.

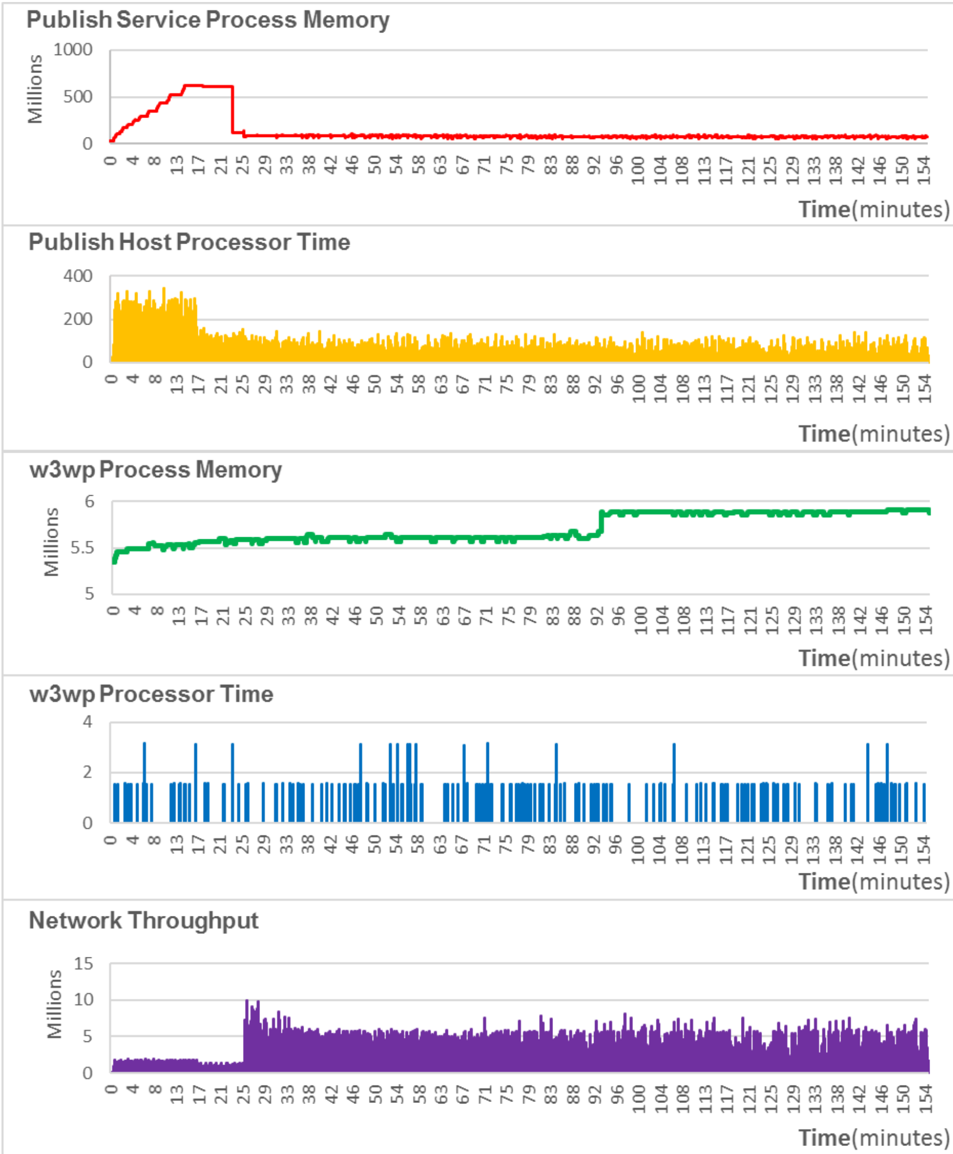
Key Metrics



This chart displays a comparison between the time that was spent publishing items using the Publishing Service and the standard Sitecore publishing functionality. The Publishing Service decreases the publishing time for content items and media items by 10 and 3.7 times respectively.

All the publish operations were run using the Publish Item (Include subitems) option.

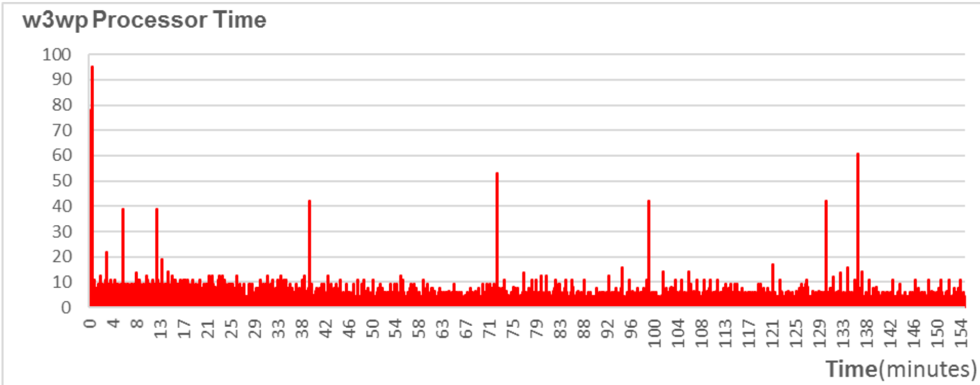
Publish Content Items: Publish Host

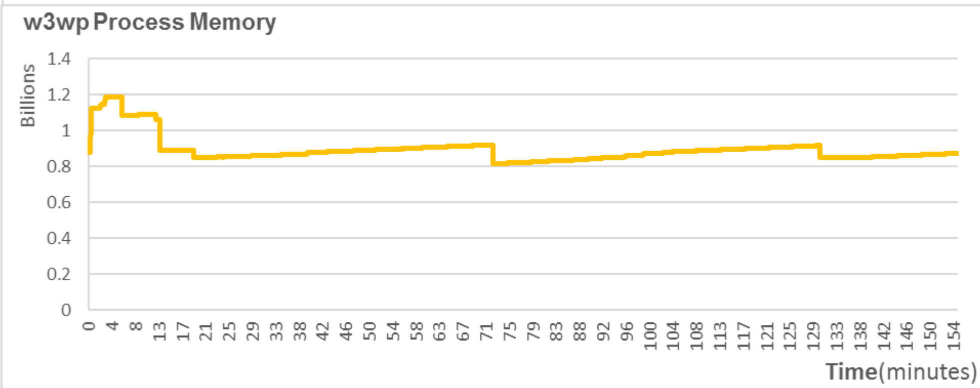


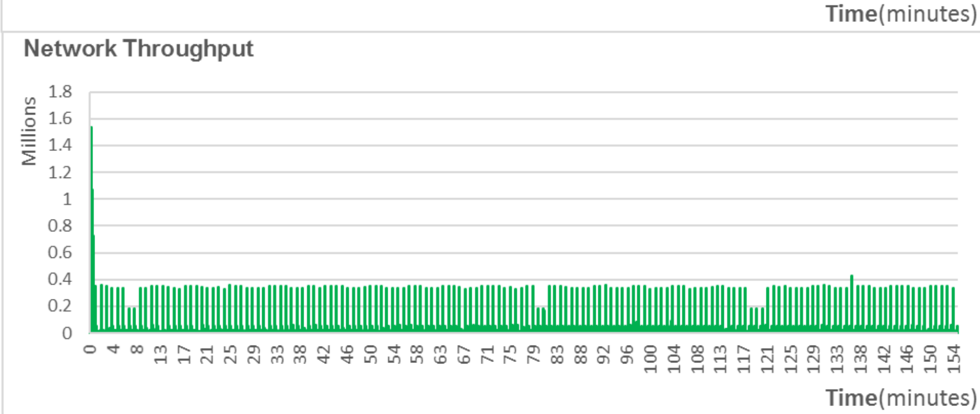
Memory usage in the main process (`Sitecore.Framework.Publishing.Host`) takes the maximum, about 620 MB, in the beginning (during manifest calculation) and remains near 80MB during the rest part of the publishing process.

The main process (`Sitecore.Framework.Publishing.Host`) takes 75% of processor time in the beginning and takes about 25% of processor time during the rest of the publishing process.

Publish Content Items: Sitecore Server

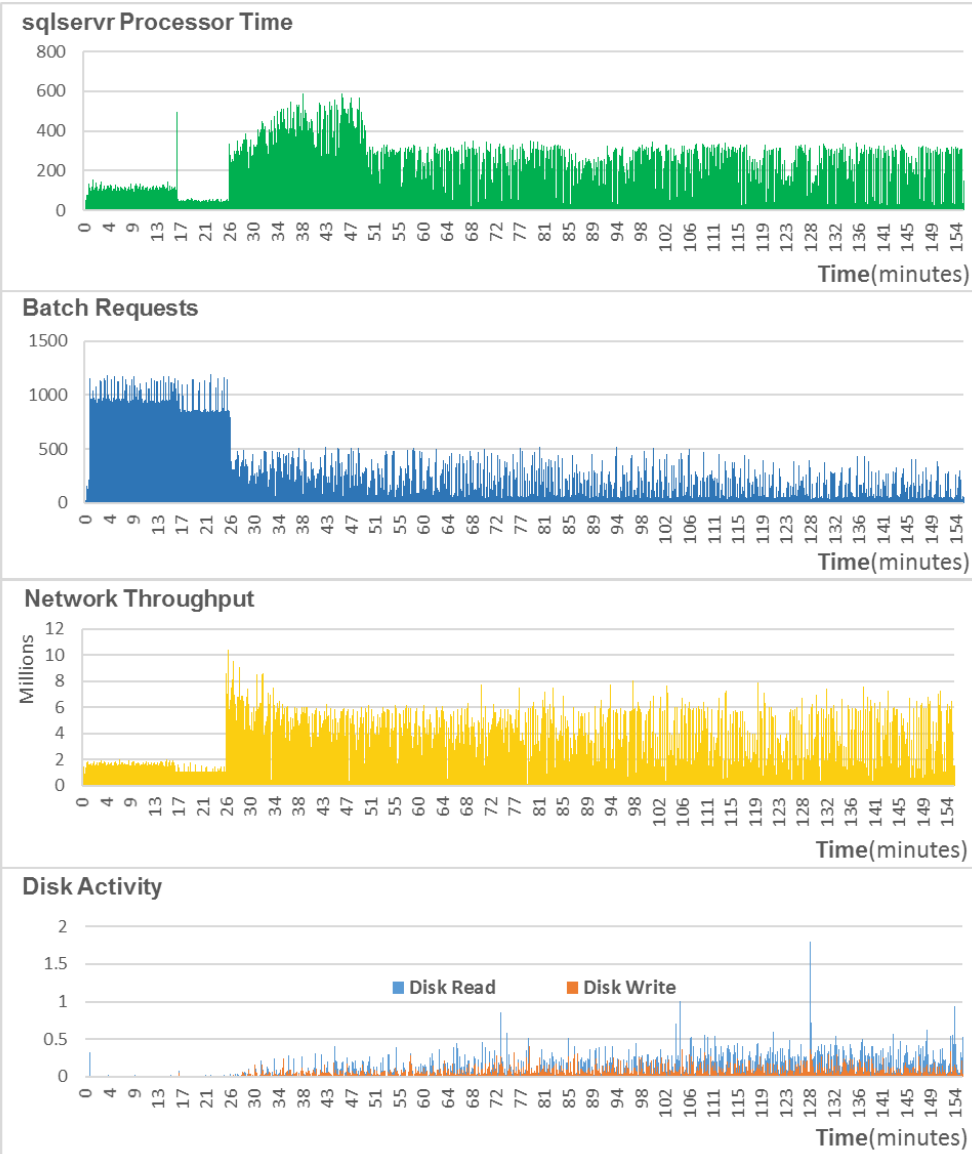






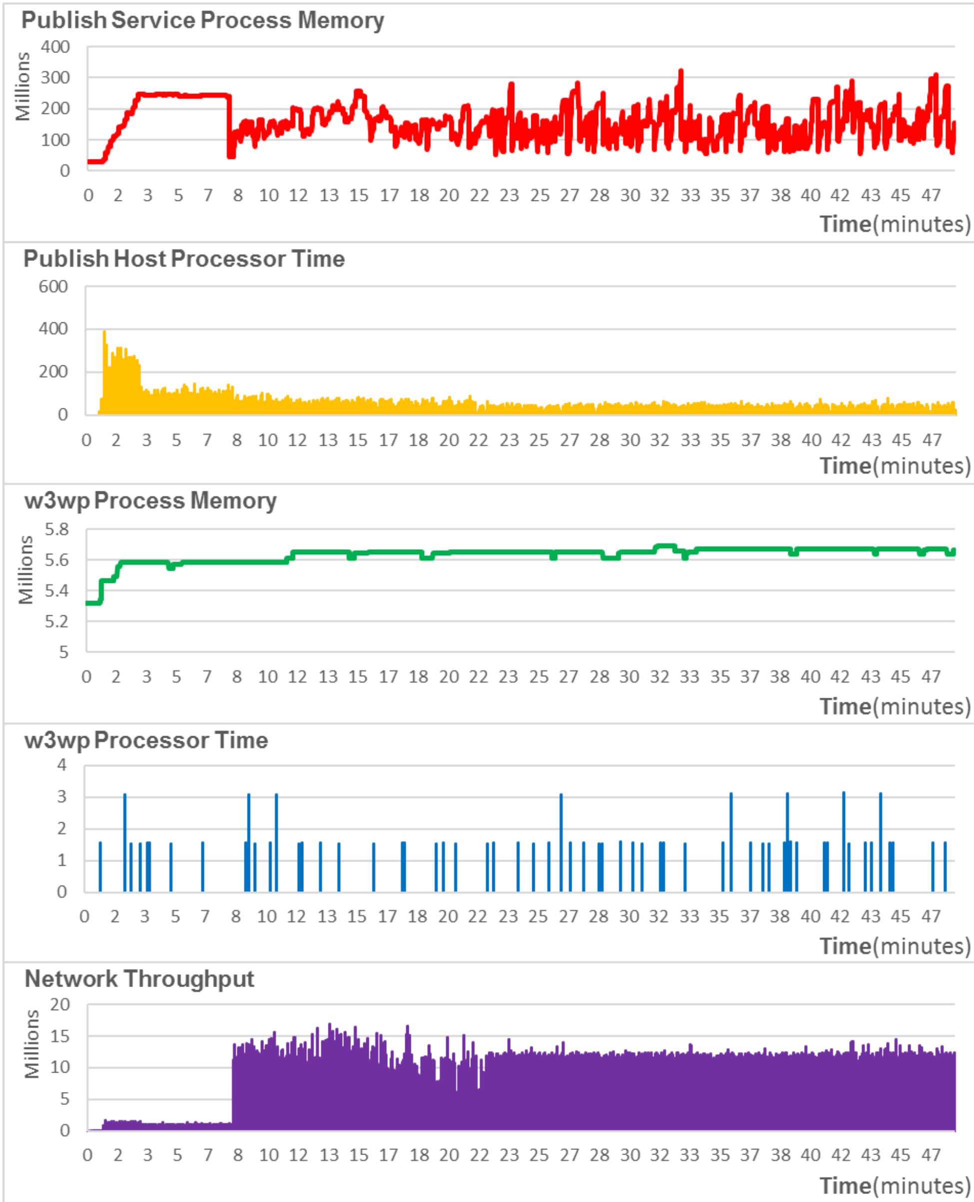
The publishing process does not have any considerable impact on Sitecore instance.

Publish Content Items: SQL Server



In the beginning, during manifest calculation, SQL server receives many Transact-SQL batch requests. Nevertheless, it does not have a huge impact on the network or the processor. During the remainder of the publishing process, all the parameters sustain a stable load level that is well within the SQL server capabilities.

Publish Media Items: Publish Host



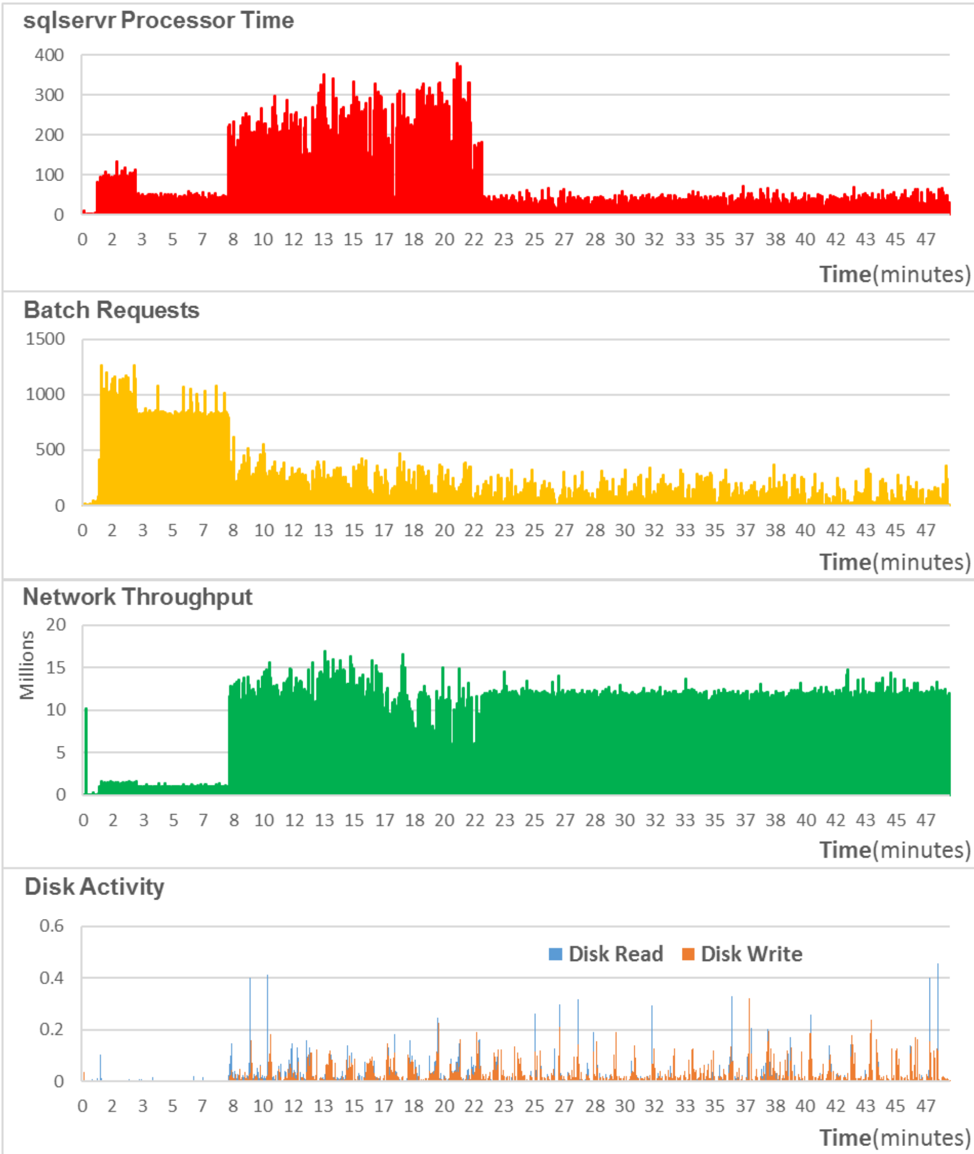
Memory usage in the main process (`Sitecore.Framework.Publishing.Host`) remains between 50 and 325 MB during the whole publishing process.

The main process (`Sitecore.Framework.Publishing.Host`) takes 75% of processor time in the beginning and takes about 25% of processor time during the rest of the publishing process.

Publish Media Items: Sitecore Server

The publishing process does not have any considerable impact on the Sitecore instance.

Publish Media Items: SQL Server



In the beginning, during manifest calculation, SQL server receives many Transact-SQL batch requests. Nevertheless, this does not have a huge impact on the network or the processor.

During the remainder of the publishing process, all parameters sustain a stable load level that is well within SQL server's capabilities.

Appendix

Session Database Performance Script

```
USE [Sitecore_Sessions];
GO

BEGIN TRANSACTION;
GO

IF( OBJECT_ID( N'dbo.sessions', N'SN' ) IS NOT NULL )
BEGIN
  DROP SYNONYM [dbo].[Sessions];
END;
GO

CREATE SYNONYM [dbo].[Sessions] FOR [tempdb].[dbo].[SessionState];
GO

IF( EXISTS( SELECT 1 FROM [information_schema].[tables] WHERE ([table_schema] = 'dbo') AND
([table_type] = 'BASE TABLE') AND ([table_name] = 'SessionState') ) )
BEGIN
  DROP TABLE [dbo].[SessionState];
END;

IF( OBJECT_ID( N'dbo.applications', N'SN' ) IS NOT NULL )
BEGIN
  DROP SYNONYM [dbo].[Applications];
END;
GO

CREATE SYNONYM [dbo].[Applications] FOR [tempdb].[dbo].[Application];
GO

IF( EXISTS( SELECT 1 FROM [information_schema].[tables] WHERE ([table_schema] = 'dbo') AND
([table_type] = 'BASE TABLE') AND ([table_name] = 'Application') ) )
BEGIN
  DROP TABLE [dbo].[Application];
END;

COMMIT TRANSACTION;
GO



USE [master];

BEGIN TRANSACTION;
GO

IF( OBJECT_ID( N'dbo.Sitecore_InitializeSessionState', N'P') IS NOT NULL )
BEGIN
  DROP PROCEDURE [dbo].[Sitecore_InitializeSessionState];
END;
GO

CREATE PROCEDURE [dbo].[Sitecore_InitializeSessionState] AS
BEGIN
  EXECUTE [Sitecore_Sessions].[dbo].[CreateTables];
END;
GO
```

```
EXECUTE [dbo].[Sitecore_InitializeSessionState];
GO

COMMIT TRANSACTION;
GO

EXECUTE [sp_procoption] @ProcName = 'dbo.Sitecore_InitializeSessionState', @OptionName =
'startup', @OptionValue = 'true';
GO
```