

# Performance White Paper

## Sitecore Email Experience Manager 9.2

*Sitecore EXM 9.2 Performance Testing*

## Table of Contents

Executive summary.....	4
EXM overview.....	5
Managing email campaigns .....	5
Sending emails.....	6
Rendering emails.....	6
Tracking emails .....	7
EXM Performance Testing Approach.....	8
Phase I. Sending emails .....	8
Phase II. Tracking emails .....	10
Load profile .....	10
Deployment.....	11
Azure configuration .....	11
Configuration and scaling.....	11
Test scenarios .....	12
Test infrastructure configuration .....	13
Monitoring configuration.....	13
Performance metrics .....	13
Results.....	14
Phase I. Sending emails .....	14
Test summary.....	14
EXM Dispatch Metrics.....	15
CPU – App service plan .....	16
App memory.....	17
Connections.....	18
Azure SQL databases – DTU percentage .....	19
Azure Search.....	20
Azure Cache - Redis .....	20
Azure Service Bus .....	21
Phase II. Tracking email throughput.....	22
Test summary.....	22
Response times – pages/requests.....	23
Requests per second .....	23
Response times – CD, XC-Search, XC-Collect, REP, RefData .....	23
CPU – app service plan.....	24

App memory.....	25
Azure SQL databases .....	26
Azure Search.....	27
Azure Cache - Redis .....	28
Azure Service Bus .....	28

*Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2019 Sitecore. All rights reserved.*

## Executive summary

This white paper describes Sitecore Email Experience Manager (EXM) performance testing.

EXM is one of the main parts of the Sitecore Experience platform. It allows you to create individual email campaigns and make them both personal and relevant for all clients.

The objectives of EXM performance testing are to:

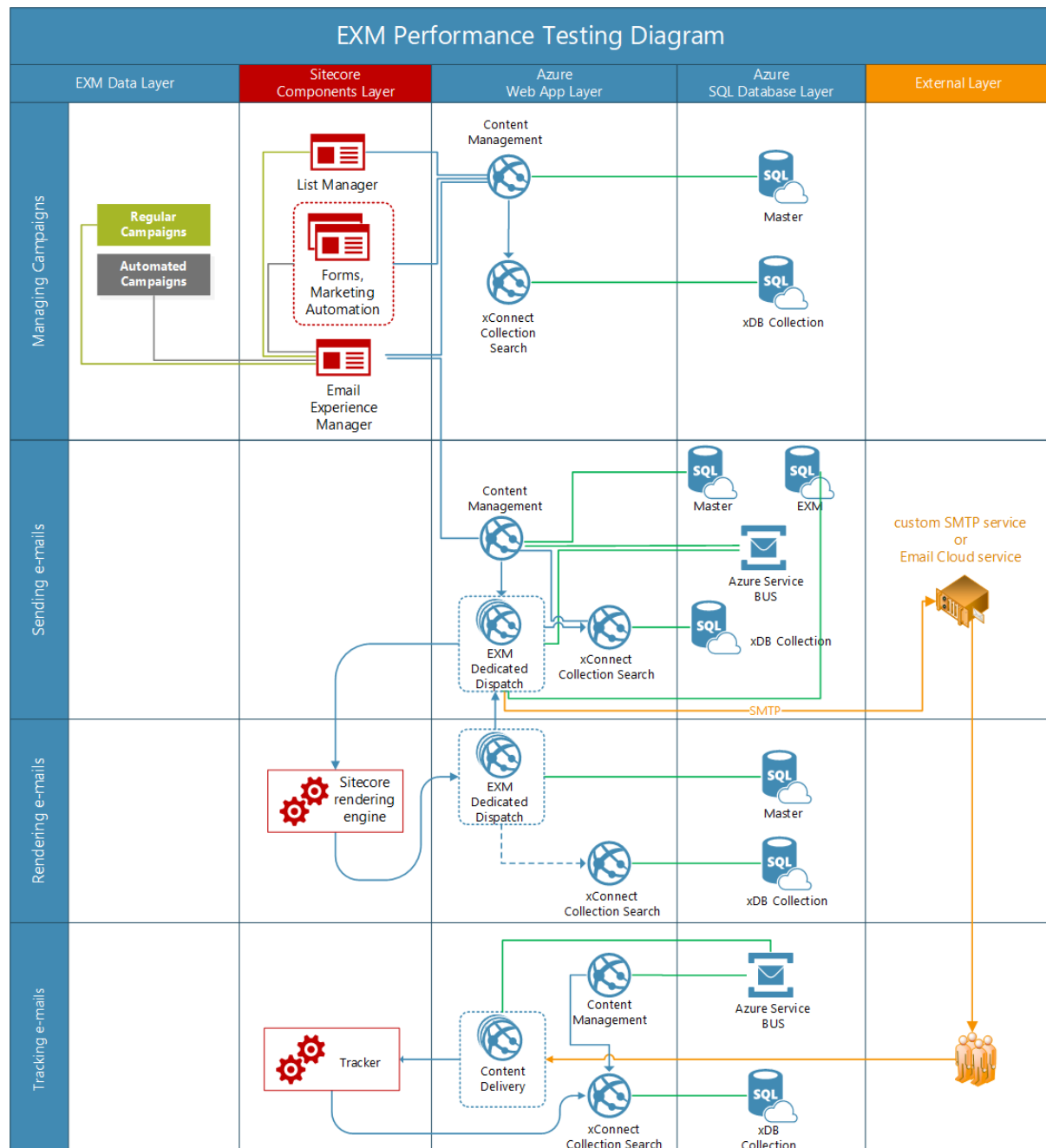
- Check the ability of the system to operate at the expected load levels.
- Establish a baseline performance based on an “Extra Large” deployment.
- Identify possible bottlenecks in the system.

The tests focus on the following three main domains:

- Content Management
- Dedicated EXM server
- Content Delivery

## EXM overview

EXM is an integrated part of the Sitecore Experience Platform that relies heavily on the [tracking](#), [reporting](#), and segmentation functionality of the Sitecore Experience Database (xDB). You can also use the email campaign capabilities if you use Sitecore Experience Manager to manage and personalize emails, as well as deliver the individually personalized messages to your customers.



### Managing email campaigns

You manage email campaigns in the EXM application that is served by the Content Management role. The campaigns are stored in the Master database.

There are two types of campaigns available in EXM:

- Email campaigns – the email campaigns associated with contact lists.
- Automated email campaigns – used in the Marketing Automation and Forms applications. EXM sends emails to individual contacts when they trigger goals or events on your website. You can also send emails programmatically using the Client API.

You can associate email campaigns with contact lists, or segments that you manage through the List Manager. In the List Manager, there are two types of list: contact lists and segmented lists. Contact lists target a specific group of contacts. A segmented list filters an existing contact list. Both types of list are stored in the xDB. The List Manager API uses the xConnect Collection Search role to retrieve contacts in lists.

You can associate an email campaign with multiple contact lists or segmented lists. You can also use lists to select the contacts to include or exclude in the dispatch process.

## Sending emails

The [EXM Dispatch](#) role sends a specific email message to a contact. Depending on the configuration, you can use a [custom SMTP service](#) or the Email Cloud service using SMTP to send the message.

The EXM Dispatch role then saves an Email Sent interaction to the [xConnect Collection Search](#) role. This records the email campaign on the individual contact in the xDB, provides reporting in EXM and [Experience Analytics](#), and enables [personalization](#) across other channels.

The EXM Dispatch role then removes the contact from the dispatch queue in the EXM database.

During the dispatch job, the Content Management role waits for job completion on all EXM Dispatch roles. When all EXM Dispatch roles are done and all contacts in the EXM database queue have been processed, the Content Management role changes the state of the email campaign in the Master database to *Sent*.

## Rendering emails

The EXM Dispatch role renders an email message by sending a HTTP request and generates a page through the Sitecore rendering engine. The email message contains content items that come from the Master database.

For simple, personalized emails that use basic token replacement, the EXM Dispatch role generates the email message once and caches it for all contacts. For highly personalized emails, a HTTP request is sent to the EXM Dispatch role and emails render separately for each contact. Personalization therefore puts a significantly higher load on the EXM Dispatch role, which makes scaling considerations important.

For reporting purposes, the EXM Dispatch role saves a message on the Message Bus to update the email address history facet.

The Content Management role handles the message and updates the contact through the xConnect Collection Search role.

## Tracking emails

After EXM sends an email message and the recipient opens it, the Content Delivery role receives a request. The request returns an empty 1x1 pixel.

The Content Delivery role handles the request and stores an *Open event* message in the Message Bus.

The Content Management role handles this message and stores an interaction for the contact that contains an Open event in xConnect Collection role. An EXM-specific xConnect plugin runs on the xConnect Collection Search role to update the specific contact facets on the contact that Sitecore uses for reporting purposes.

All the links in the email campaigns are associated with a specific EXM tracking page. When a recipient clicks an email link, the tracking page uses the standard tracker to create an interaction for the contact and registers a *Click page* event. The tracking page then redirects to the actual page in the link.

The tracker uses the normal tracking and collections data flows. When a session ends, the Content Delivery role sends the interaction to the xConnect Collection role. An EXM-specific plugin runs on the xConnect Collection role to update specific facets for reporting purposes.

# EXM Performance Testing Approach

The purpose of testing is to make sure that by Sitecore 9.2 EXM can send 25 million messages per month, and also to identify possible bottlenecks in the system.

In consideration of the business logic and features of working with email companies, the EXM performance testing is divided into two phases:

## Phase I. Sending emails

In this phase, we have combined the following processes:

- Managing email campaigns
- Rendering emails
- Sending emails

The sending emails process uses third-party solutions to send emails (SMTP server, Email Cloud services), and we therefore decided to use a stub to bypass third-party applications.

EXM lets you test campaign throughput by emulating a message transfer agent (MTA). MTA emulation lets you mimic the round-trip time required to send an email from the Sitecore CMS to the MTA. For more information, see the article [Testing EXM performance in emulation mode](#).

Unfortunately, this solution has some limitations:

- Emulation sending mode does not move the message to the *Sent* state – the message status changes back to *Drafts* after the process has completed.
- Emulation sending mode is not available for message variants when you run an A/B test.
- The most critical limitation for performance testing is that when you use this solution, most of the email sending pipeline remains unused –as well as the marketing automation, processing, and reporting pipelines.



# Performance White Paper

The unused methods are highlighted in red in the following screenshots:

```

-0.12% MessageTaskRunner worker thread 1 - -36,121 ms
-0.12% ThreadStart - -36,121 ms - System.Threading.ThreadHelper.ThreadStart
-0.12% Run - -36,121 ms - System.Threading.ExecutionContext.Run(ExecutionContext, ContextCallback, Object)
-0.12% Run - -36,121 ms - System.Threading.ExecutionContext.Run(ExecutionContext, ContextCallback, Object, Boolean)
-0.12% RunInternal - -36,121 ms - System.Threading.ExecutionContext.RunInternal(ExecutionContext, ContextCallback, Object, Boolean)
-0.12% WorkerMain - -36,121 ms - Sitecore.EmailCampaign.Cm.Dispatch.MessageTaskRunner.WorkerMain
-0.12% SendToNextRecipient - -36,121 ms - Sitecore.EmailCampaign.Cm.Dispatch.MessageTask.SendToNextRecipient
-0.12% OnSendToNextRecipient - -36,123 ms - Sitecore.EmailCampaign.Cm.Dispatch.DispatchTask.OnSendToNextRecipient
-0.16% RunPipeline - -46,180 ms - Sitecore.Modules.EmailCampaign.Core.PipelineHelper.RunPipeline(String, PipelineArgs)
-0.16% Run - -46,180 ms - Sitecore.Pipelines.DefaultCorePipelineManager.Run(String, PipelineArgs, String)
-0.16% Run - -46,180 ms - Sitecore.Pipelines.DefaultCorePipelineManager.Run(String, PipelineArgs, String, Boolean)
-0.16% Run - -46,180 ms - Sitecore.Pipelines.CorePipeline.Run(PipelineArgs)
-0.16% Process - -46,228 ms - Sitecore.EmailCampaign.Cm.Pipelines.SendEmail.SendEmail.Process(SendMessageArgs)
-0.86% SendAsync - +251,282 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchManager.SendAsync(EmailMessage)
-0.86% Start - +251,281 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.Start(ref TStateMachine)
-0.86% MoveNext - +251,279 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchManager.SendAsync<d_6.MoveNext>
-0.86% SendAsync - +251,277 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchProviderBase.SendAsync(EmailMessage)
-0.86% Start - +251,276 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.Start(ref TStateMachine)
-0.86% MoveNext - +251,274 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchProviderBase.SendAsync<d_5.MoveNext>
-0.86% SendEmailAsync - +251,236 ms - +1,000 calls - Sitecore.EDS.Providers.CustomSmtplib.DispatchProvider.SendEmailAsync(EmailMessage)
-0.86% Start - +251,235 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.Start(ref TStateMachine)
-0.86% MoveNext - +251,232 ms - +1,000 calls - Sitecore.EDS.Providers.CustomSmtplib.DispatchProvider.SendEmailAsync<d_3.MoveNext>
-0.86% SendAsync - +250,649 ms - +1,000 calls - Sitecore.EDS.Core.NetSmtplib.ChilkatMessageTransport.SendAsync(TransportClient)
-0.00% ChilkatMessageTransport ctor - +544 ms - +1,000 calls - Sitecore.EDS.Core.NetSmtplib.ChilkatMessageTransport.ctor(EmailMessage)
-0.00% GetConnectionAsync - +10 ms - +1,000 calls - Sitecore.EDS.Core.NetSmtplib.ChilkatConnectionPool.GetConnectionAsync
-0.00% BaseAdd - +8 ms - +2,000 calls - System.Collections.Specialized.NameValueCollectionBase.BaseAdd(String, Object)
-0.00% GetTimestamp - +6 ms - +2,000 calls - System.Diagnostics.Stopwatch.GetTimestamp
-0.00% Add - +4 ms - +2,000 calls - System.Collections.Specialized.NameValueCollection.Add(String, String)
-0.00% Stop - +2 ms - +2,000 calls - System.Diagnostics.Stopwatch.Stop
-0.00% get_CurrentInfo - +1 ms - +2,000 calls - System.Globalization.NumberFormatInfo.get_CurrentInfo
-0.00% get_Statistics - +1 ms - +2,000 calls - Sitecore.EDS.Core.Dispatch.DispatchResult.get_Statistics
-0.00% GetElapsedDateTimeTicks - +1 ms - +2,000 calls - System.Diagnostics.Stopwatch.GetElapsedDateTimeTicks
-0.00% SetResult - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.SetResult(TResult)
-0.00% Undo - 0 ms - +1,000 calls - System.Threading.ExecutionContextSwitcher.Undo
-0.00% EstablishCopyOnWriteScope - 0 ms - +1,000 calls - System.Threading.ExecutionContext.EstablishCopyOnWriteScope(Thread, Boolean, ref ExecutionContextSwitcher)
-0.00% get_Task - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.get_Task
-0.00% SetMessageHeader - +29 ms - +1,000 calls - Sitecore.EDS.Providers.CustomSmtplib.DispatchProvider.SetMessageHeader(EmailMessage)
-0.00% BaseAdd - +4 ms - +1,000 calls - System.Collections.Specialized.NameValueCollectionBase.BaseAdd(String, Object)
-0.00% BaseSet - +2 ms - +1,000 calls - System.Collections.Specialized.NameValueCollectionBase.BaseSet(String, Object)
-0.00% GetAwaiter - 0 ms - +1,000 calls - System.Threading.Tasks.Task`1.GetAwaiter
-0.00% Set - 0 ms - +1,000 calls - System.Collections.Specialized.NameValueCollection.Set(String, String)
-0.00% SetResult - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.SetResult(TResult)
-0.00% ArgumentNotNull - 0 ms - +1,000 calls - Sitecore.Diagnostics.Assert.ArgumentNotNull(Object, String)
-0.00% GetResult - 0 ms - +1,000 calls - System.Runtime.CompilerServices.TaskAwaiter`1.GetResult
-0.00% get_IsCompleted - 0 ms - +1,000 calls - System.Runtime.CompilerServices.TaskAwaiter`1.get_IsCompleted
-0.00% Undo - 0 ms - +1,000 calls - System.Threading.ExecutionContextSwitcher.Undo
-0.00% EstablishCopyOnWriteScope - 0 ms - +1,000 calls - System.Threading.ExecutionContext.EstablishCopyOnWriteScope(Thread, Boolean, ref ExecutionContextSwitcher)
-0.00% get_Task - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.get_Task
-0.00% DispatchProviderBase<SendAsync>d_5.ctor - 0 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchProviderBase.SendAsync<d_5.ctor>
-0.00% Create - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.Create
-0.00% get_Provider - 0 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchManager.get_Provider
-0.00% SetResult - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.SetResult(TResult)
-0.00% GetResult - 0 ms - +1,000 calls - System.Runtime.CompilerServices.TaskAwaiter`1.GetResult

+0.00% GetAwaiter - 0 ms - +1,000 calls - System.Threading.Tasks.Task`1.GetAwaiter
+0.00% get_IsCompleted - 0 ms - +1,000 calls - System.Runtime.CompilerServices.TaskAwaiter`1.get_IsCompleted
+0.00% Undo - 0 ms - +1,000 calls - System.Threading.ExecutionContextSwitcher.Undo
+0.00% EstablishCopyOnWriteScope - 0 ms - +1,000 calls - System.Threading.ExecutionContext.EstablishCopyOnWriteScope(Thread, Boolean, ref ExecutionContextSwitcher)
+0.00% DispatchManager<SendAsync>d_6.ctor - 0 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.DispatchManager.SendAsync<d_6.ctor>
+0.00% get_Task - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.get_Task
+0.00% Create - 0 ms - +1,000 calls - System.Runtime.CompilerServices.AsyncTaskMethodBuilder`1.Create
+0.00% Format - 0 ms - System.String.Format(String, params Object[])
+0.00% get_IncludePllnLogFiles - -3 ms - Sitecore.EmailCampaign.Model.Web.Settings.GlobalSettings.get_IncludePllnLogFiles
+0.00% ParseInt32 - +4 ms - +4,000 calls - System.Number.ParseInt32(String, NumberStyles, NumberFormatInfo)
+0.00% AddMessage - -3 ms - Sitecore.Pipelines.PipelineArgs.AddMessage(String, PipelineMessageType)
+0.00% get_CurrentInfo - +2 ms - +4,000 calls - System.Globalization.NumberFormatInfo.get_CurrentInfo
+0.00% GetAsOneString - 0 ms - +4,000 calls - System.Collections.Specialized.NameValueCollection.GetAsOneString(ArrayList)
+0.00% get_Item - 0 ms - Sitecore.Collections.SafeDictionary`2.get_Item(Key)
+0.00% get_IsConfigured - +1 ms - Sitecore.EDS.Core.Dispatch.DispatchManager.get_IsConfigured
+0.00% Get - +1 ms - +4,000 calls - System.Collections.Specialized.NameValueCollection.Get(Int32)
+0.00% get_Result - +1 ms - +1,000 calls - System.Threading.Tasks.Task`1.get_Result
+0.00% FormatWith - 0 ms - Sitecore.StringExtensions.StringExtensions.FormatWith(String, params Object[])
+0.00% AddSendingTime - +1 ms - +1,000 calls - Sitecore.Modules.EmailCampaign.Core.Dispatch.TimeSummary.AddSendingTime(Int32)
+0.00% GetTimeDiff - 0 ms - Sitecore.Modules.EmailCampaign.Util.GetTimeDiff(DateTime, DateTime)
+0.00% AddMailParsingTime - 0 ms - +1,000 calls - Sitecore.Modules.EmailCampaign.Core.Dispatch.TimeSummary.AddMailParsingTime(Int32)
+0.00% get_Summary - 0 ms - +4,000 calls - Sitecore.EmailCampaign.Cm.Dispatch.MessageTask.get_Summary
+0.00% get_Emulation - 0 ms - Sitecore.Modules.EmailCampaign.Messages.MessageItem.get_Emulation
+0.00% get_Statistics - 0 ms - +4,000 calls - Sitecore.EDS.Core.Dispatch.DispatchResult.get_Statistics
+0.00% AddGetConnectionTime - 0 ms - +1,000 calls - Sitecore.Modules.EmailCampaign.Core.Dispatch.TimeSummary.AddGetConnectionTime(Int32)
+0.00% get_UtcNow - -1 ms - System.DateTime.get_UtcNow
+0.00% get_MtaEmulation - 0 ms - Sitecore.EmailCampaign.Model.Web.Settings.GlobalSettings.get_MtaEmulation
+0.00% get_Subject - 0 ms - +2,000 calls - Sitecore.EDS.Core.Dispatch.EmailMessage.get_Subject
+0.00% TracerInfo - 0 ms - Sitecore.ExM.Framework.Diagnostics.Logger.TracerInfo(String)
+0.00% AddMailSizeSend - 0 ms - +1,000 calls - Sitecore.Modules.EmailCampaign.Core.Dispatch.TimeSummary.AddMailSizeSend(Int32)
+0.00% get_Message - 0 ms - Sitecore.EmailCampaign.Cm.Dispatch.MessageTask.get_Message
+0.00% get_CustomData - 0 ms - Sitecore.Pipelines.PipelineArgs.get_CustomData
+0.00% get_Recipients - 0 ms - +1,000 calls - Sitecore.EDS.Core.Dispatch.EmailMessage.get_Recipients
+0.00% ArgumentNotNull - 0 ms - Sitecore.Diagnostics.Assert.ArgumentNotNull(Object, String)
+0.00% get_EmulationMinSendTime - 0 ms - +1,000 calls - Sitecore.EmailCampaign.Model.Web.Settings.GlobalSettings.get_EmulationMinSendTime
+0.00% get_IsConfigured - 0 ms - +1,000 calls - Sitecore.EDS.Core.ProviderInitializer`1.get_IsConfigured
+1.02% Sleep - -297,511 ms - +1,000 calls - System.Threading.Thread.Sleep(Int32)
+0.00% get_EmulationFailProbability - -1 ms - +1,000 calls - Sitecore.EmailCampaign.Model.Web.Settings.GlobalSettings.get_EmulationFailProbability
+0.00% Next - -3 ms - +2,000 calls - System.Random.Next(Int32, Int32)
+0.00% get_EmulationMaxSendTime - 0 ms - +1,000 calls - Sitecore.EmailCampaign.Model.Web.Settings.GlobalSettings.get_EmulationMaxSendTime
+0.00% FillEmail - +145 ms - Sitecore.EmailCampaign.Cm.Pipelines.SendEmail.FillEmail.Process(SendMessageArgs)
+0.00% WaitOne - -99 ms - System.Threading.WaitHandle.WaitOne(Int32, Boolean)
+0.00% Process - +3 ms - Sitecore.EmailCampaign.Cm.Pipelines.SendEmail.FillEmail.Process(SendMessageArgs)
+0.00% Increment - 0 ms - Sitecore.Diagnostics.PerformanceCounter.AmountPerSecondCounter.Increment(Int64)
+0.00% GetMethod - 0 ms - Sitecore.Pipelines.CoreProcessor.GetMethod(Object[])
+0.00% Dispose - 0 ms - Sitecore.Diagnostics.Profiling.NullPipelineProfilingScope.Dispose
+0.00% Invoke - 0 ms - Sitecore.Pipelines.PipelineMethod.Invoke(Object)
+0.00% Process - 0 ms - Sitecore.EmailCampaign.Cm.Pipelines.SendEmail.Sleep.Process(SendMessageArgs)
+0.00% StartProcessorScope - 0 ms - Sitecore.Diagnostics.Profiling.NullPipelineProfilingScope.StartProcessorScope(Int32)
+0.00% RegisterProcessor - 0 ms - Sitecore.Diagnostics.Profiling.NullPipelineProfilingScope.RegisterProcessor(Int32, String, String)
+0.00% StartPipelineScope - 0 ms - Sitecore.Diagnostics.Profiling.ProfilerApi.StartPipelineScope(String, Int32)
+0.00% StartPipelineProfilingScope - 0 ms - Sitecore.Diagnostics.Profiling.NullPipelineProfiler.StartPipelineProfilingScope(String, Int32)
+0.00% WaitOne - 0 ms - System.Threading.WaitHandle.WaitOne(Int32)
+0.00% ArgumentNotNull - 0 ms - Sitecore.Diagnostics.Assert.ArgumentNotNull(Object, String)

```

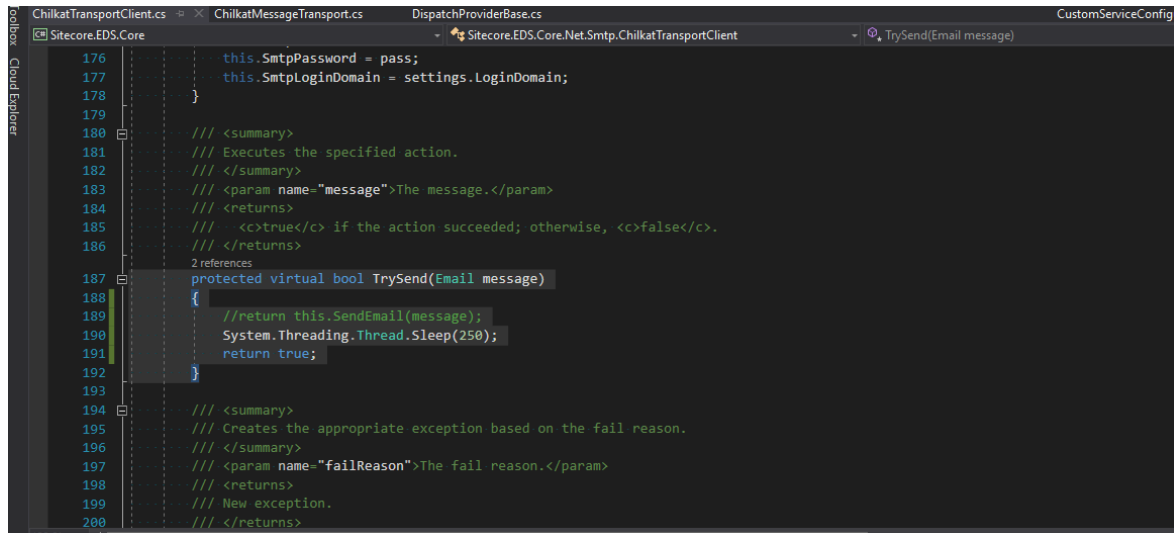
Obviously, to carry out a proper performance test of EXM, we needed to find another approach that meets performance testing requirements.

## Performance White Paper

To do this, we changed the `TrySend()` method in the `ChilkatTransportClient` class that is located in the `Sitecore.EDS.Core.Net.Smtp` namespace.

### Note

The main disadvantage of this approach is that it requires access to the source code of the EXM application. I hope that in the future, the EXM team will offer a more convenient solution.



```

176         this.SmtpPassword = pass;
177         this.SmtpLoginDomain = settings.LoginDomain;
178     }
179
180     /// <summary>
181     /// Executes the specified action.
182     /// </summary>
183     /// <param name="message">The message.</param>
184     /// <returns>
185     /// <true/> if the action succeeded; otherwise, <false/>.
186     /// </returns>
187     2 references
188     protected virtual bool TrySend(Email message)
189     {
190         //return this.SendEmail(message);
191         System.Threading.Thread.Sleep(250);
192         return true;
193     }
194
195     /// <summary>
196     /// Creates the appropriate exception based on the fail reason.
197     /// </summary>
198     /// <param name="failReason">The fail reason.</param>
199     /// <returns>
200     /// New exception.
201     /// </returns>
  
```

## Phase II. Tracking emails

All the links in the email campaigns are associated with a specific EXM tracking page. When a recipient clicks an email link, the tracking page uses the standard tracker to create an interaction for the contact and registers a *Click page* event. The tracking page then redirects to the actual page in the link.

The tracker uses the normal tracking and collections data flows. When a session ends, the Content Delivery role sends the interaction to the xConnect Collection role. An EXM-specific plugin runs on the xConnect Collection role to update specific facets for reporting purposes.

In the frame of this performance test, we tracked the following events:

- Open
- Click
- Unsubscribe
- Unsubscribe from all

## Load profile

Load	Value
Number of emails per month	25 million
Number of emails per dispatch	100K
Expected Hourly Send Rate	900K per hour
	Open: 100%

Load	Value
Number of interactions with campaign	Click: 25%
	Unsubscribe: 1%
	Unsubscribe from List: 1%
	Unsubscribe from All: 1%
Email size	50kb, 75kb, 100kb, 200 kb, 400 kb

## Deployment

### Azure configuration

Azure topology	Sitecore topology configuration
XP "Extra Large"	1 x CM 8 x CD (Open & Click Handling) 1 x xConnect Search 2 x xConnect Collection 1 x xConnect Ref 1 x Processing 1 x Reporting 3 x Dedicated Dispatch

### Configuration and scaling

The EXM Dispatch server does not support Azure horizontal scaling, which is why we use 3x DDS servers.

Azure SQL Databases		App Service plans		
Name	Pricing Tier	Role	Pricing Tier/Apps	Instances
core-db	Standard S1: 20 DTUs	CD-HP	S3: 8	1
exmmaster-db	Standard S2: 50 DTUs	CM-HP	S3: 1	1
forms-db	Standard S2: 50 DTUs	DDS1-HP	P3v2: 1	1
ma-db	Standard S1: 50 DTUs	DDS2-HP	P3v2: 1	1
master-db	Standard S1: 20 DTUs	DDS3-HP	P3v2: 1	1
pools-db	Standard S2: 50 DTUs	PRC-HP	S2: 1	1

Azure SQL Databases		App Service plans		
processingenginestorage-db	Standard S3: 100 DTUs	REP-HP	S2: 1	1
processingenginetasks-db	Standard S0: 10 DTUs	SI-HP	S2: 1	1
refdata-db	Standard S3: 100 DTUs	XC-Basic-HP	S3: 2	4
reporting-db	Standard S3: 100 DTUs	XC-ResourceIntensive-HP	S3: 4	3
shard0-db	Premium P4: 500 DTUs			
shard1-db	Premium P4: 500 DTUs			
smm-db	Standard S0: 10 DTUs			
tasks-db	Standard S1: 20 DTUs			
Web-db	Standard S3: 100 DTUs			

EXM Dispatch settings:

- NumberThreads = 32
- MaxGenerationThreads = 32
- DispatchEnqueueBatchSize = 300
- DispatchEnqueueThreadsNumber = 4
- EXM.DispatchBatchSize = 100

## Test scenarios

In a single marketing campaign you can use only one email template, and we need to test EXM performance for various email sizes. We must therefore repeat these scenarios for each email template.

In total, we have 5 templates for emails of 50 KB, 75 KB, 100 KB, 200 KB, and 400 KB.

Phase I

- Create a Contact List based on a file with contacts (100K Contacts).
- Create Regular Email Comparing.
- Include matching Contact List to this campaign.
- Run the campaign.

## Phase II

- Open the email.
- Click on email links according to the load profile.

## Test infrastructure configuration

### Monitoring configuration

Grafana:

- Azure Monitor Data Sources Plugin for collecting Azure Web Apps metrics
- InfluxDB for collecting metrics from JMeter

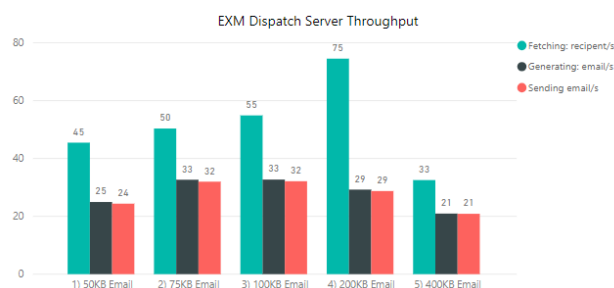
## Performance metrics

Metric	Description
SPEED	Number of emails per second during dispatch
	Number of emails per hour during dispatch
TIME SPENT	Opens processed
	Clicks processed
	Unsubscribes processed
LOAD, RESOURCE USAGE	CPU cores usage
	RAM usage
	SQL load per DB
	SQL DBs sizes
	Azure Search metrics
OTHER	Warnings, errors logged during dispatch

# Results

## Phase I. Sending emails

### Test summary



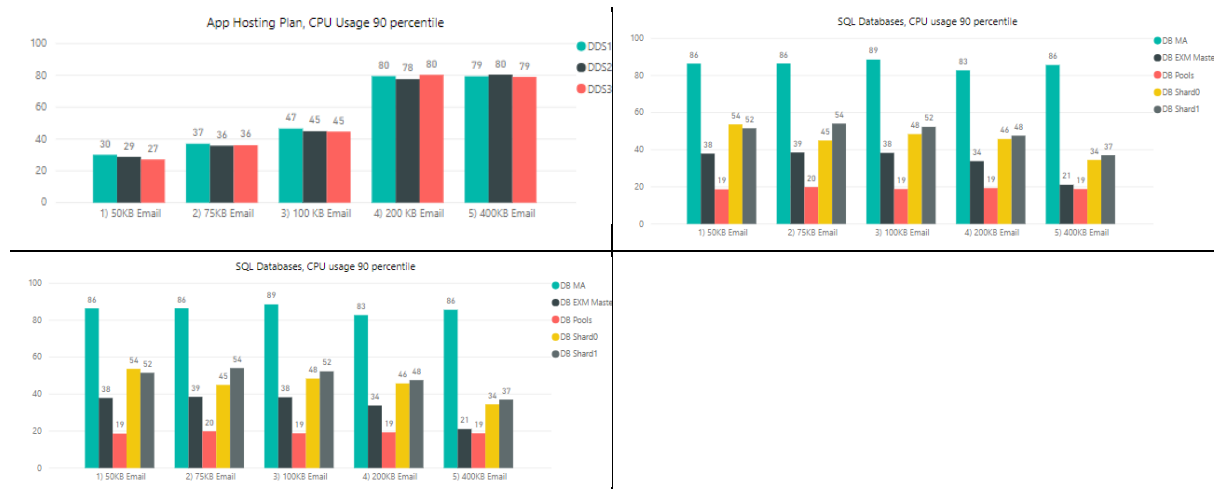
The average bandwidth capacity of the EXM dedicated dispatch server ranges from 21 to 32 emails per second. Thus, with 3 DDS, the throughput is increased to 96 emails per second, which is equal to 345 600 messages per hour.

CPU consumption of DDS servers increases from 30% up to 80% depending on the email size.

Larger emails increase CPU consumption:

E-Mail size/CPU %	50kb	75kb	100kb	200 kb	400 kb
App HP DDS1	30	37	47	80	79
App HP DDS2	29	36	45	78	80
App HP DDS3	27	36	45	80	79
App HP XC Basic	82	78	74	74	69
App HP XC Resource Intensive	6	6	6	6	7
SQL DB MA	86	86	89	83	86
SQL DB EXM Master	38	39	38	34	21
SQL DB Shard0	54	45	48	44	34
SQL DB Shard1	52	54	52	48	37

## Performance White Paper



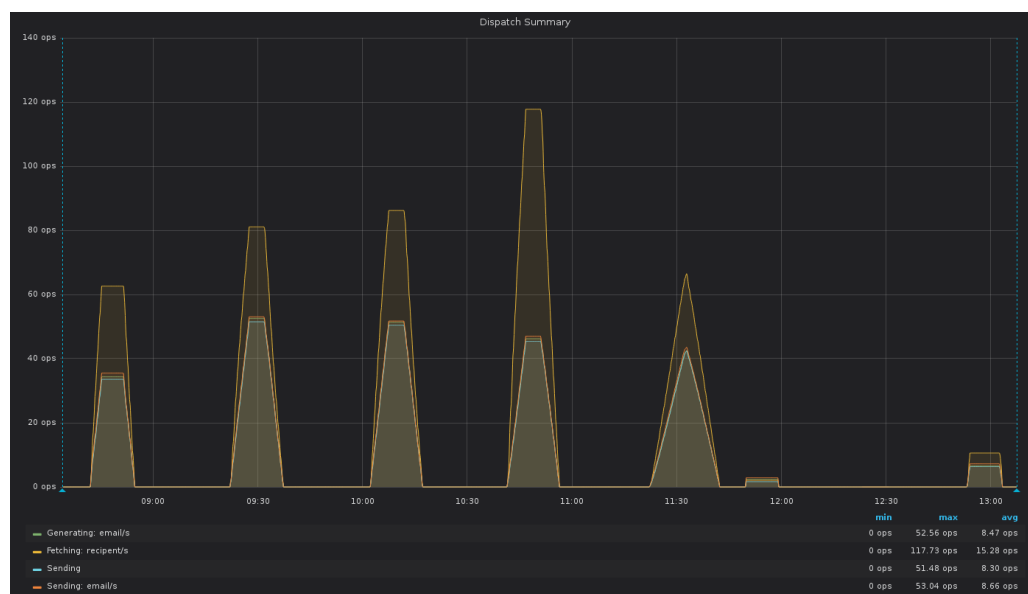
Let us suppose that we spend about 8 hours a day sending out marketing campaigns. Usually for this action the time interval with the lowest user activity should be chosen. Thus, we can process  $345.6 \text{ K} * 8 \text{ hours} = 2.764 \text{ M}$  emails per day. This means that we can complete a marketing campaign for 25 million contacts in 9 days.

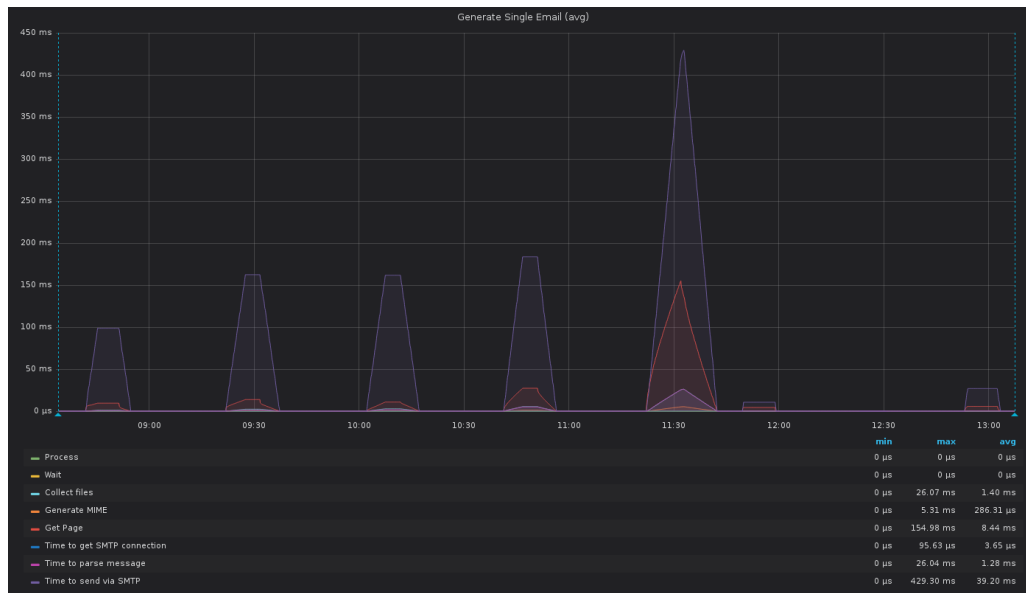
If you spend about 8 hours a day sending out marketing campaigns, you should find the time slot with the lowest level of activity.

We can process  $345.6 \text{ K} * 8 \text{ hours} = 2.764 \text{ M}$  emails per day.

It will therefore take 9 days to complete a marketing campaign for 25 million contacts.

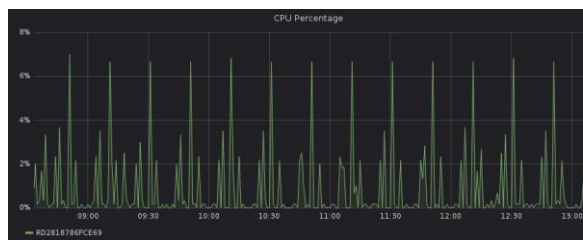
## EXM Dispatch Metrics



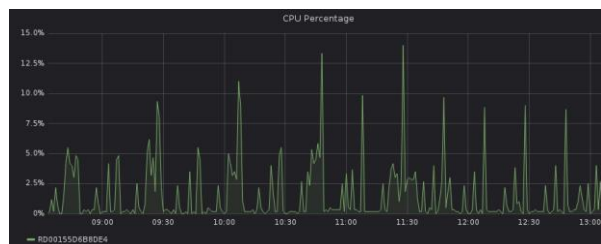


## CPU – App service plan

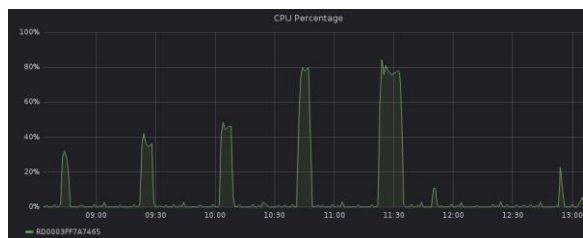
CD



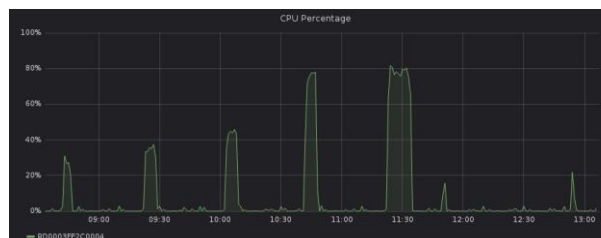
CM



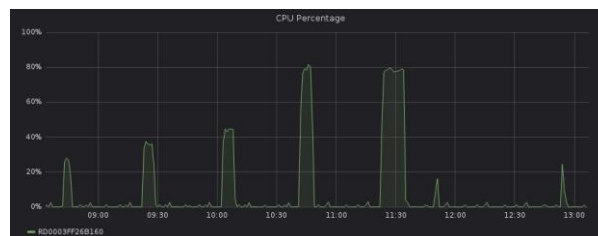
DDS1



DDS2

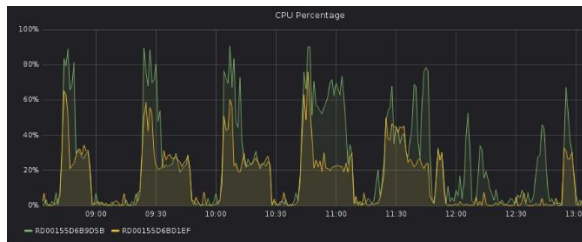


DDS3

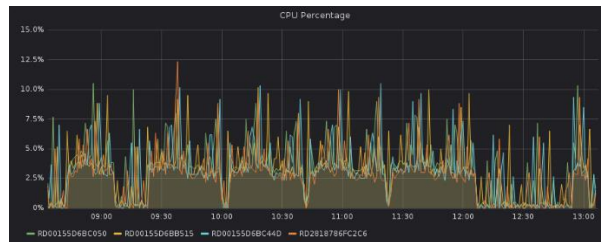




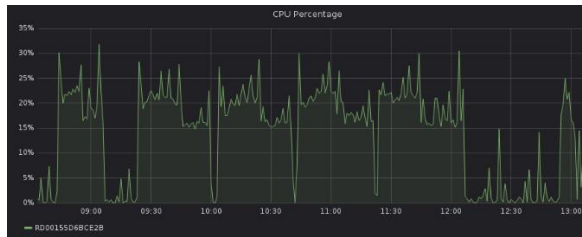
xc-Basic



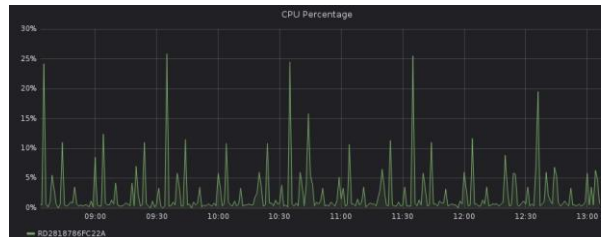
xc-ResourceIntensive



Prc

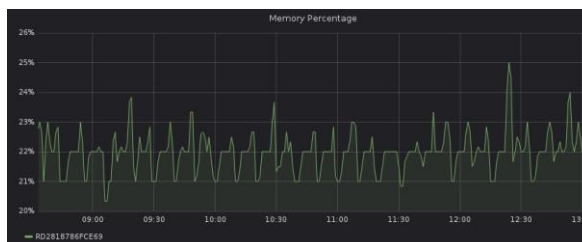


Rep

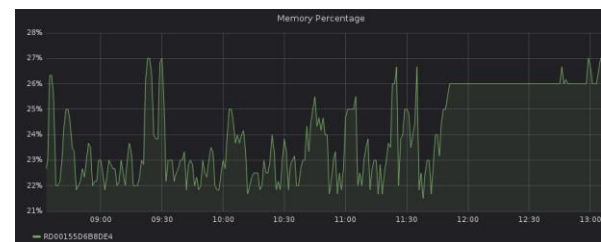


## App memory

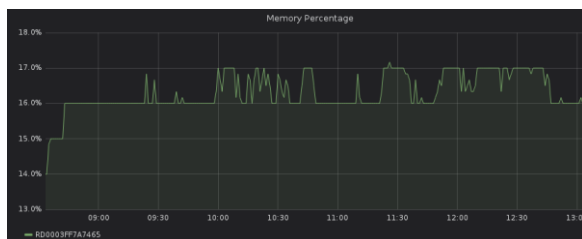
CD



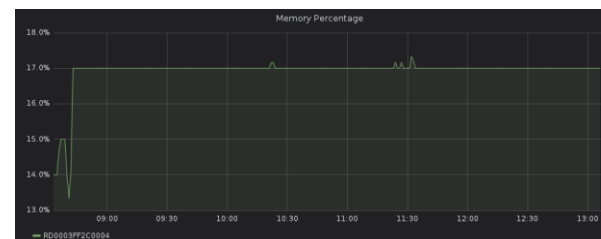
CM



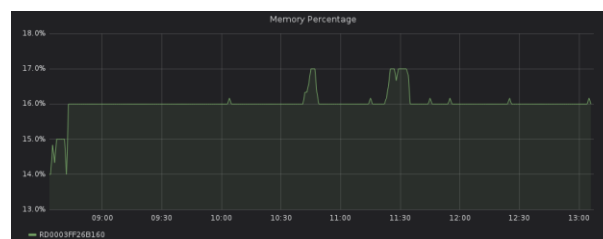
DDS1



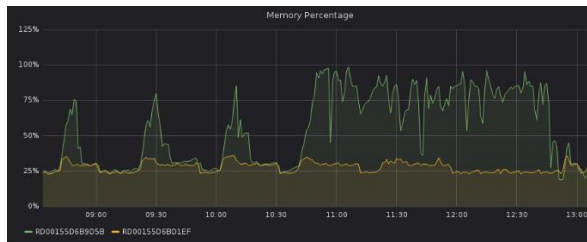
DDS2



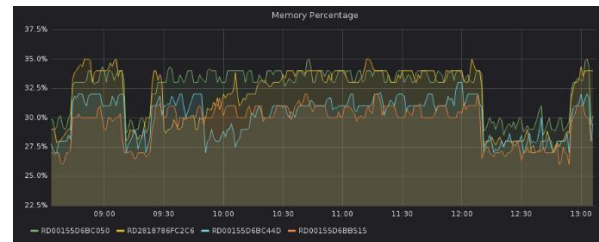
DDS3



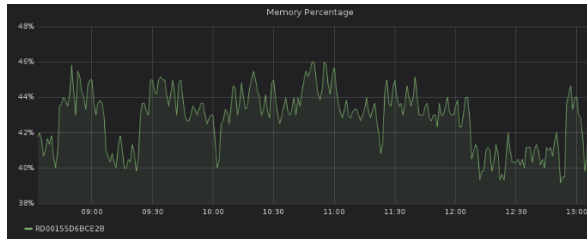
xc-Basic



xc-ResourceIntensive



Prc



Rep

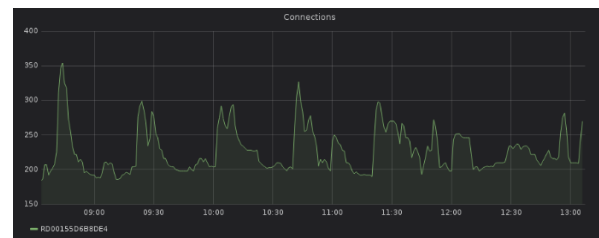


## Connections

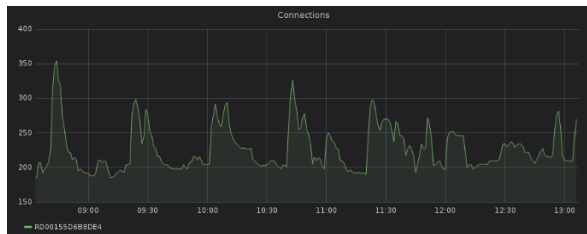
CD



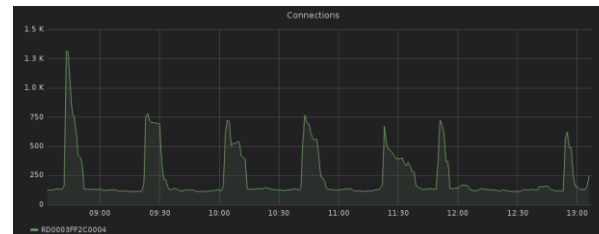
CM



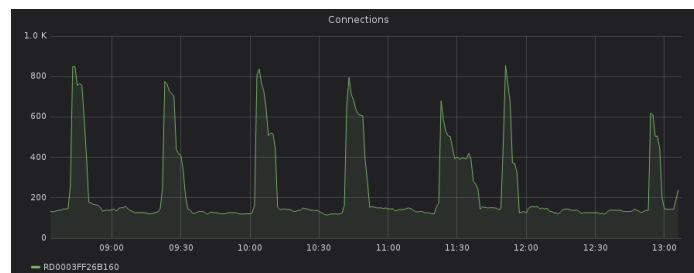
DDS1



DDS2



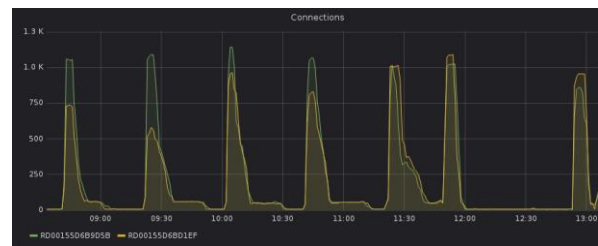
DDS3



xc-Collection



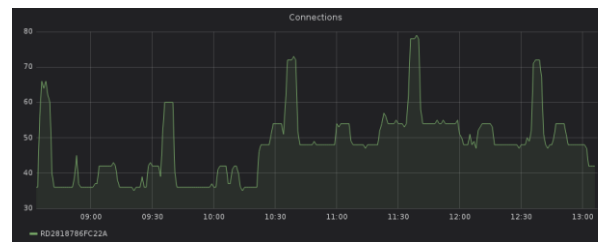
xc-search



Prc

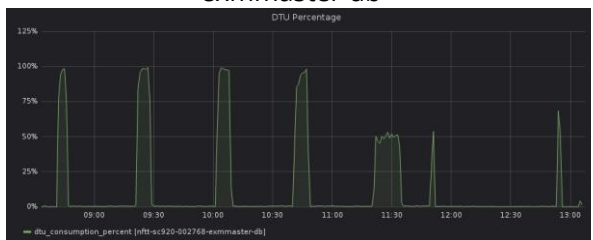


Rep



## Azure SQL databases – DTU percentage

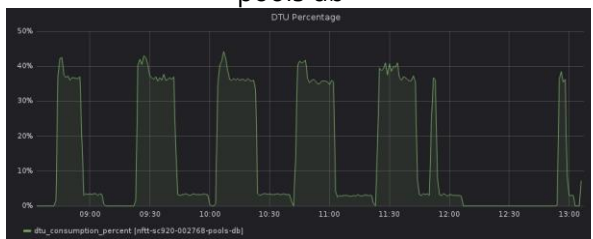
exmmaster db



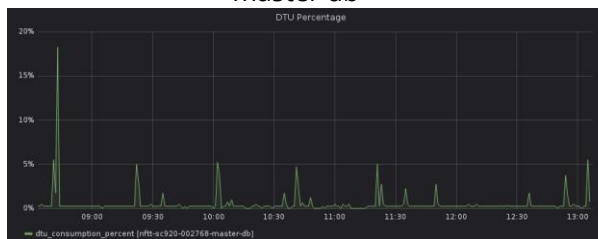
ma db



pools db



master db



shard-0 db

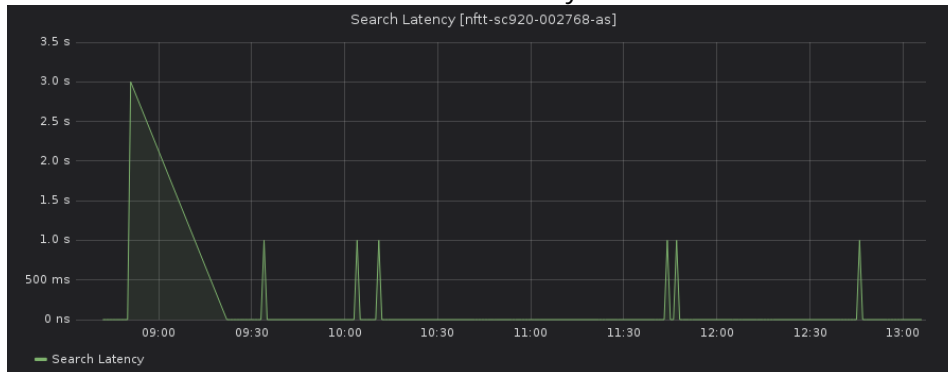


shard-1 db

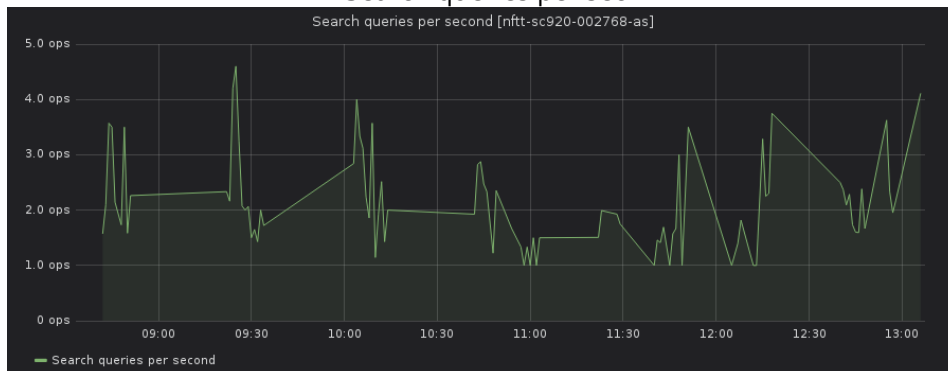


## Azure Search

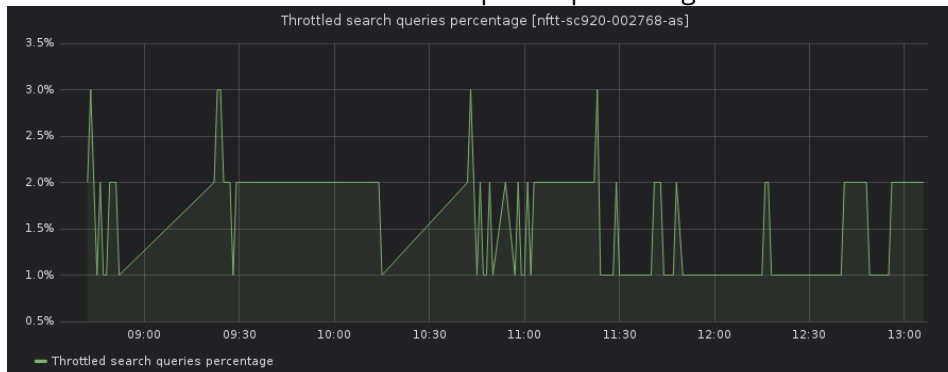
Search Latency



Search queries per sec



Throttled search queries percentage



## Azure Cache - Redis

Avg Cache Hits&Misses

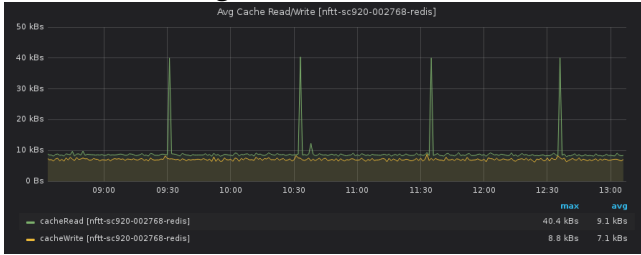


Avg Cache Latency

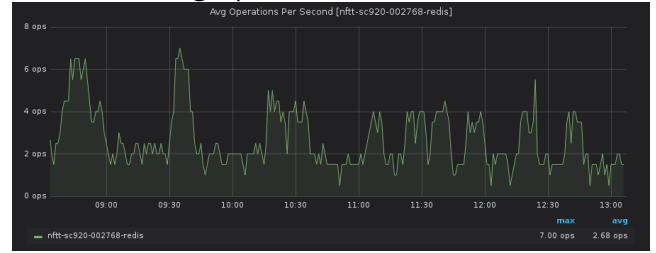


## Performance White Paper

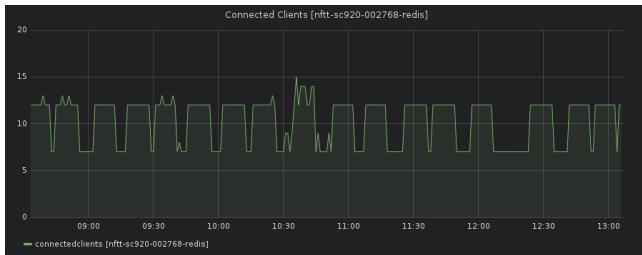
### Avg Cache Read&Write



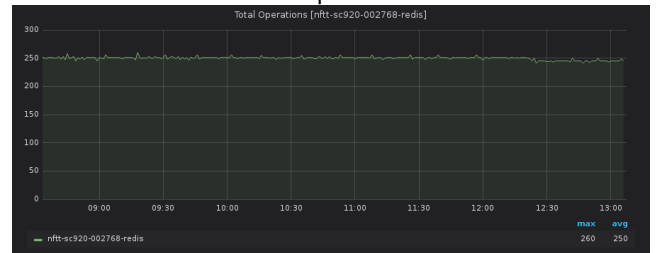
### Avg Operations Per Second



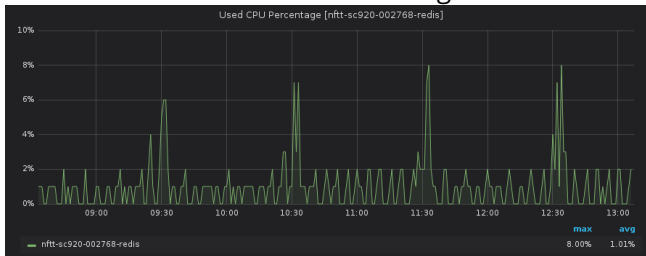
### Connected Clients



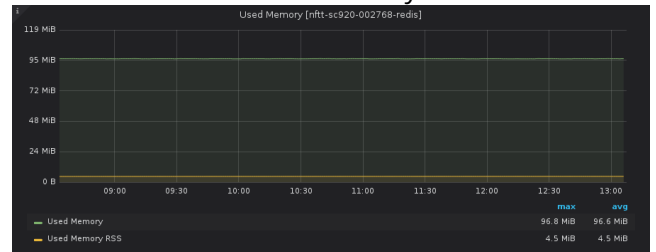
### Total Operations



### Used CPU Percentage

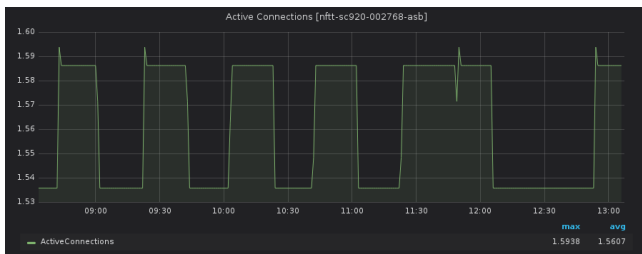


### Used Memory

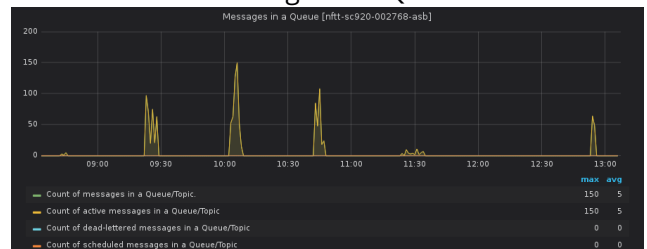


## Azure Service Bus

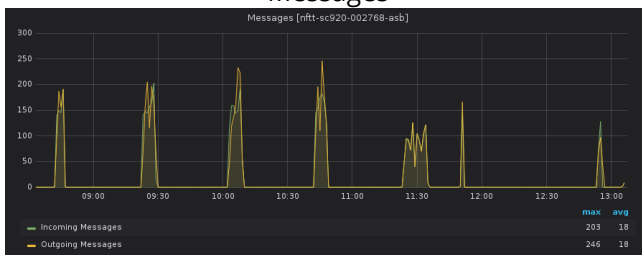
### Active Connections



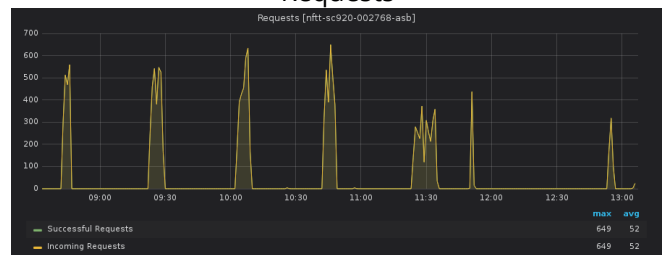
### Messages in a Queue



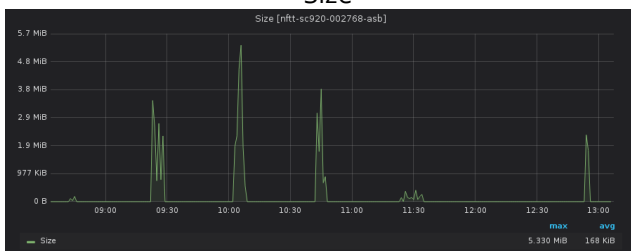
### Messages



### Requests



### Size



### Errors



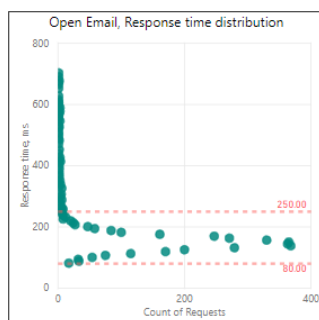
## Phase II. Tracking email throughput

### Test summary

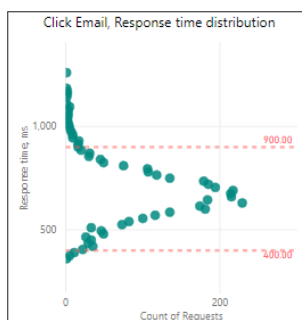


Average throughput rate during the test was about 69 requests per second, which is about 248.4 K requests per hour.

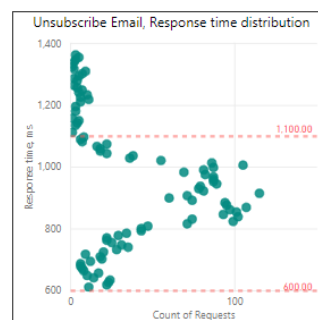
The response time for tested EXM email events is distributed from 80ms to 1200ms. More details can be seen on the following charts.



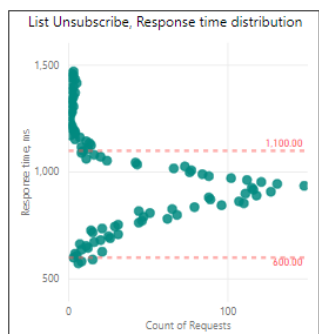
between 80–250 ms



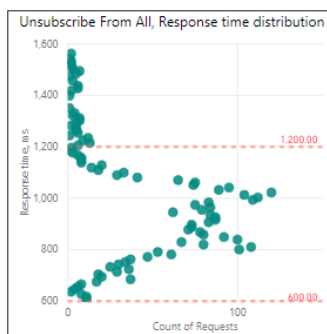
between 400 – 900 ms



between 600 – 1100 ms

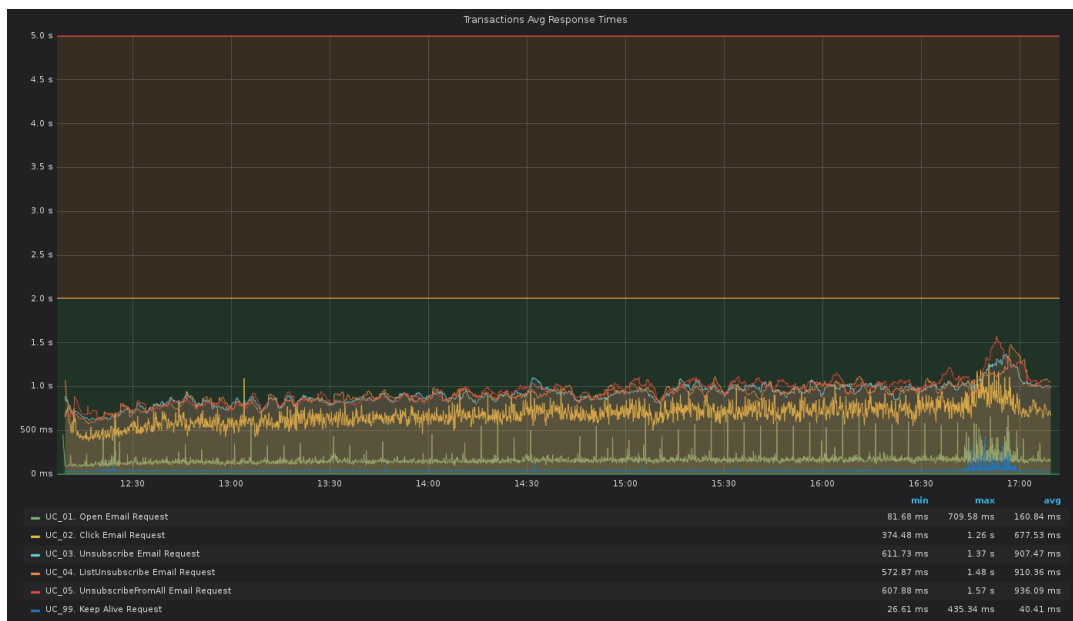


between 600 – 1100 ms

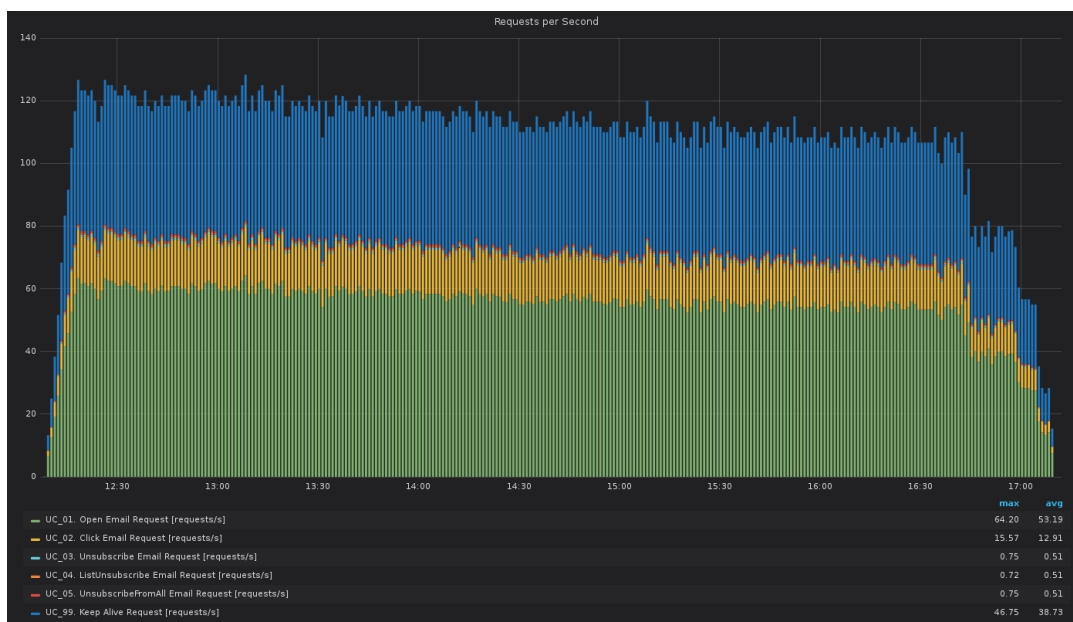


between 600 – 1200 ms

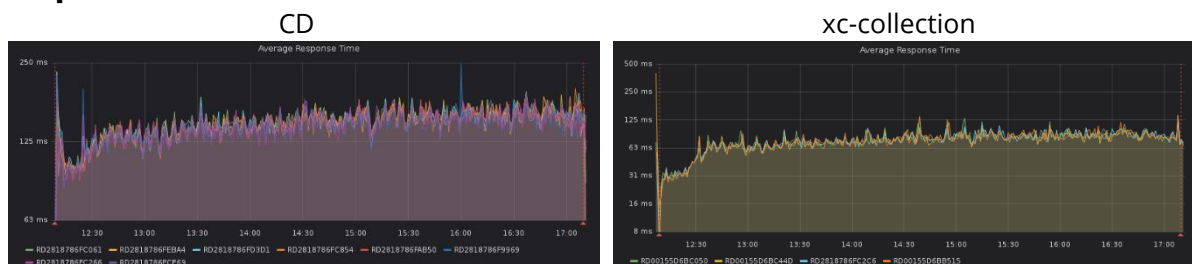
## Response times – pages/requests



## Requests per second

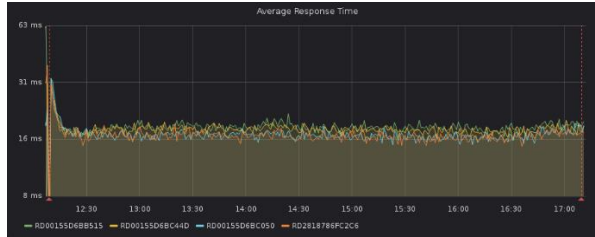


## Response times – CD, XC-Search, XC-Collect, REP, RefData

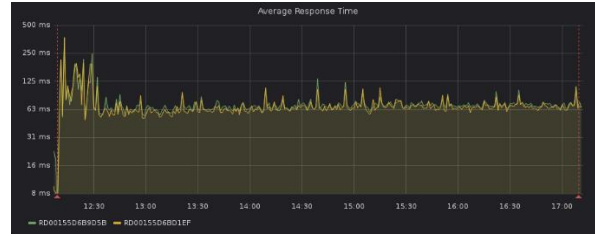




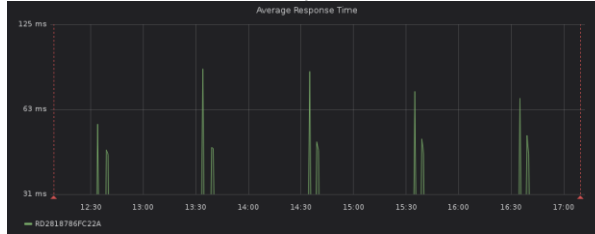
RefData



xc-search

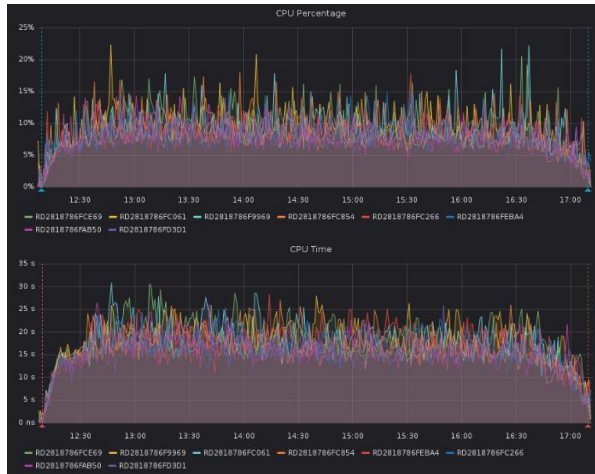


Rep



## CPU - app service plan

CD



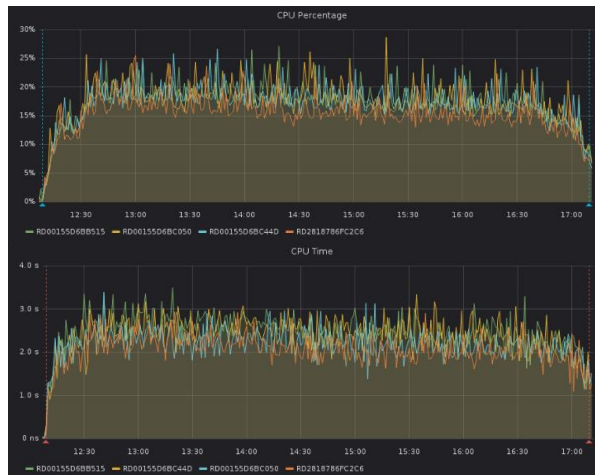
xc-Collection



xc-Search



RefData





## Marketing Automation



## Rep



## App memory

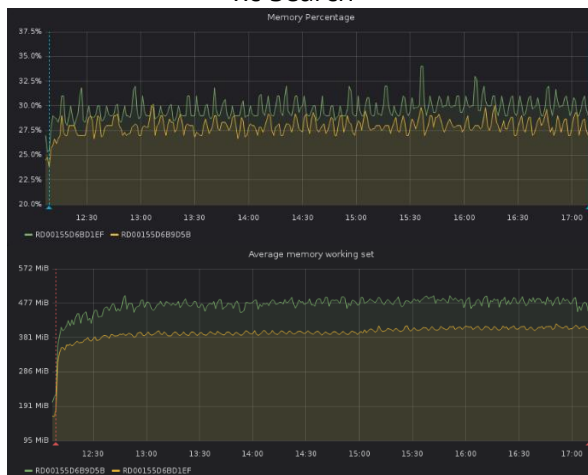
### CD



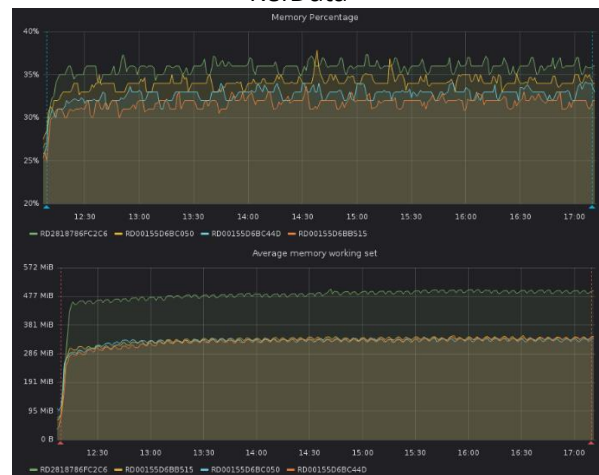
### xc-Collection



### xc-Search



### RefData



## Marketing Automation

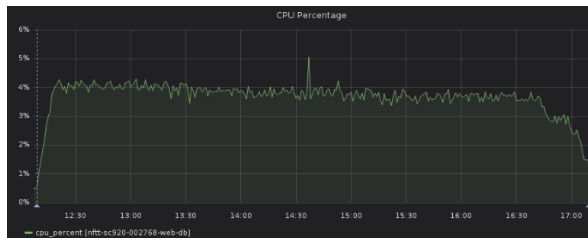


## Rep

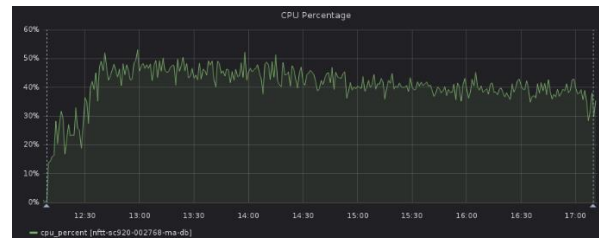


## Azure SQL databases

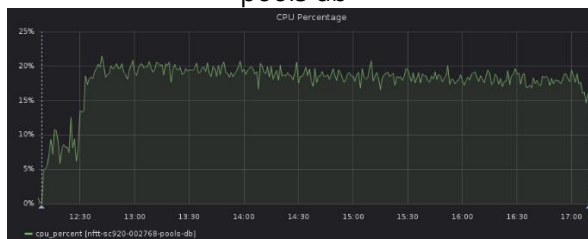
### web db



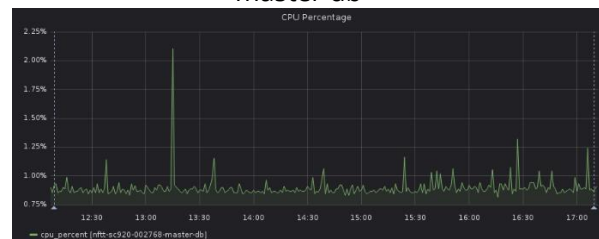
### ma db



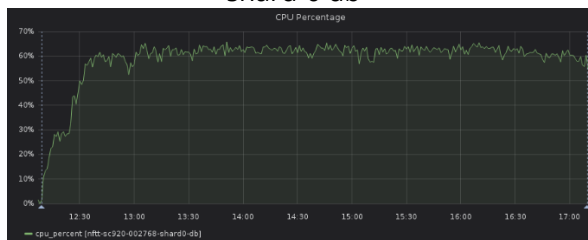
### pools db



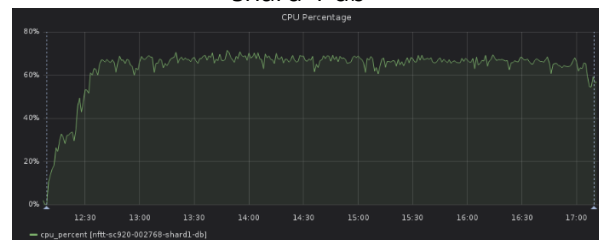
### master db



### shard-0 db



### shard-1 db

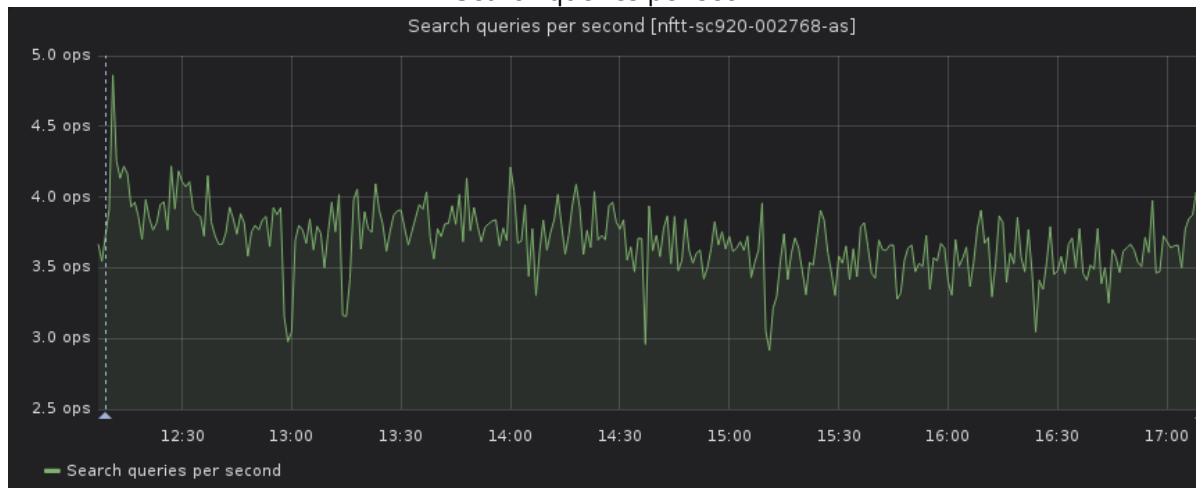


## Azure Search

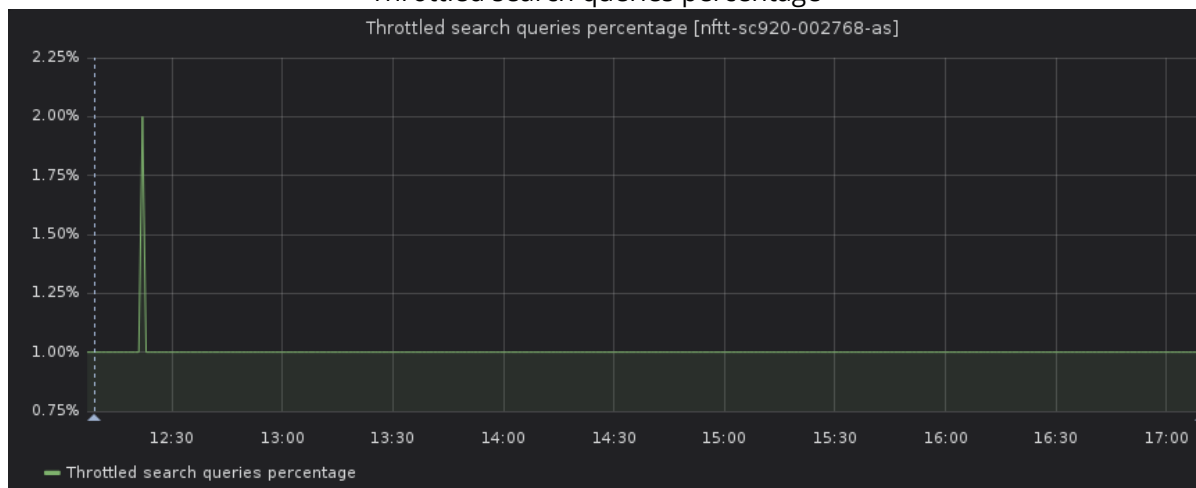
### Search Latency



### Search queries per sec

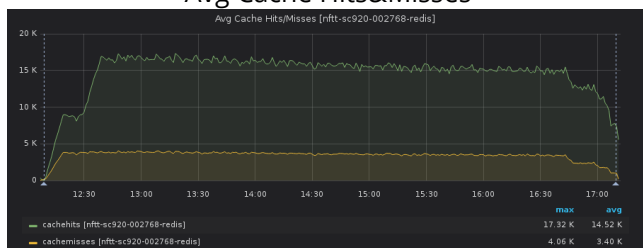


### Throttled search queries percentage

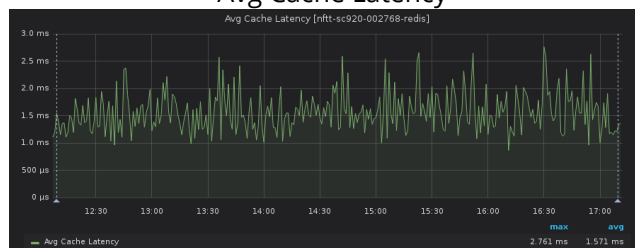


## Azure Cache - Redis

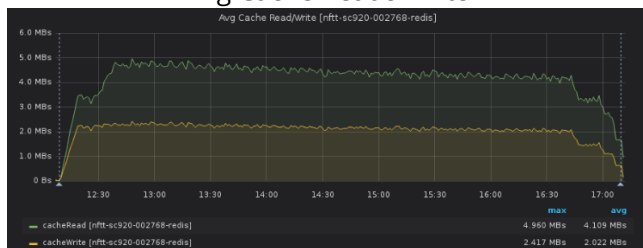
### Avg Cache Hits&Misses



### Avg Cache Latency



### Avg Cache Read&Write



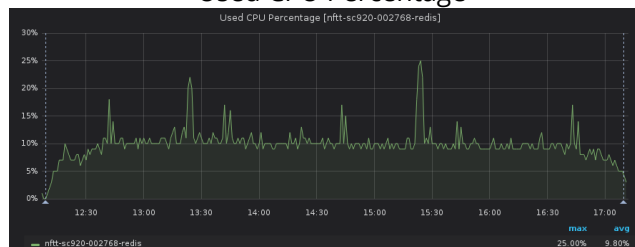
### Avg Operations Per Second



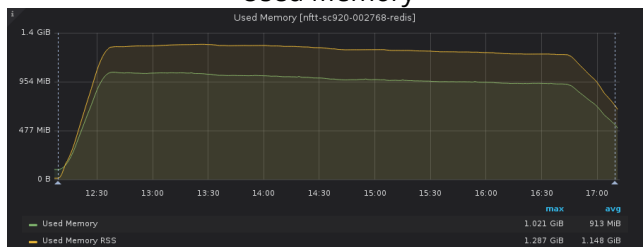
### Connected Clients



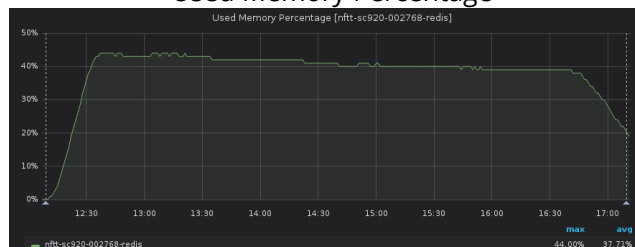
### Used CPU Percentage



### Used Memory

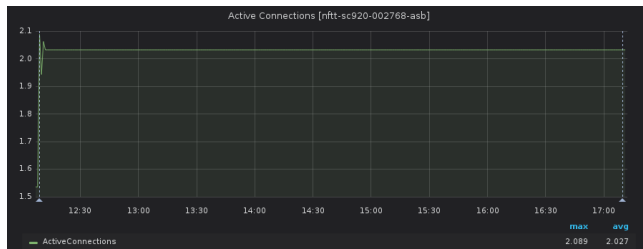


### Used Memory Percentage

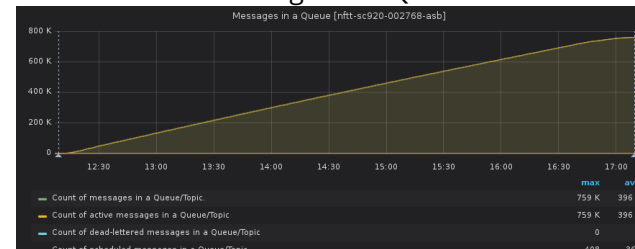


## Azure Service Bus

### Active Connections



### Messages in a Queue

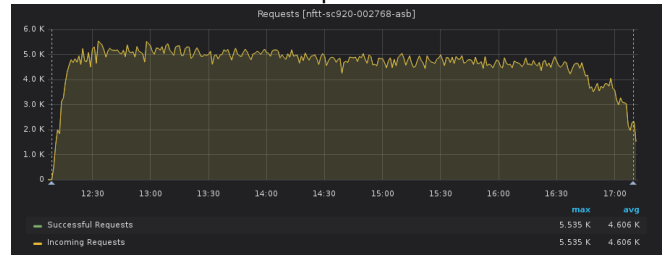


# Performance White Paper

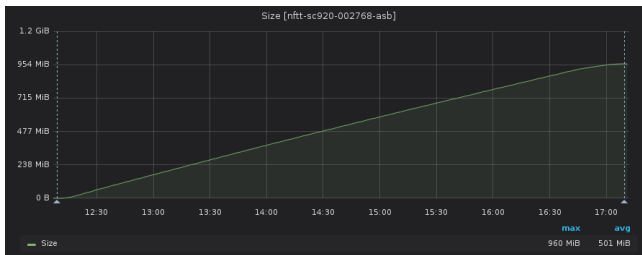
## Messages



## Requests



## Size



## Errors

