**Lab 201 – ScaleOut using MSMQ** (rev 1.2)

When the need arises to scale out a certain operation to multiple machines, after having already maxed out on the number of threads on the same machine, the Distributor feature of NServiceBus comes in handy. This feature is applicable only when using MSMQ as the transport of choice. The distributor can be thought of as a load balancer.

When scaling out, one shouldn’t have to resort to changing code. In NServiceBus, the same handler code can be made to scale out by using feature profiles: NServiceBus.Master or NServiceBus.Distributor and NServiceBus.Worker. For more information on profiles, see:

<http://particular.net/articles/more-on-profiles>

Clients are configured to send the messages to the Distributor/Master node endpoint. The distributor/master then load balances the messages it received to the worker nodes that are registered with it. As a standard NServiceBus process, the distributor maintains all the fault-tolerant and performance characteristics of NServiceBus but also is designed never to overwhelm anyof the worker nodes configured to receive work from it.

For more information see:

<http://particular.net/articles/load-balancing-with-the-distributor>

Lab Objectives

**In this lab you will learn**

* How to scale out the load on the endpoints when using MSMQ as the transport, without changing code, by simply changing configuration.
* How to set up an endpoint as either the master node or the distributor node, which will act as the load balancer.
* How to set up a node as a worker node, which will receive work from the master/distributor node.
* How the system load gets distributed by the master node to the registered worker nodes.

**NOTE**: The distributor is only applicable when using MSMQ as the message transport. It is not applicable when using other transports such as ActiveMQ, RabbitMQ, or SQLServer.

Lab Prerequisites

This lab already has these components pre-installed:

* Visual Studio 2012
* NServiceBus Infrastructure – DTC
* NServiceBus Infrastructure – MSMQ
* NServiceBus Infrastructure – RavenDB version 2.0.2261.0
* NServiceBus Infrastructure – Performance Counters

**NOTE**: To install NserviceBus infrastructure on your machines, use Powershell commandlets.

<http://particular.net/articles/managing-nservicebus-using-powershell>

Problem Definition

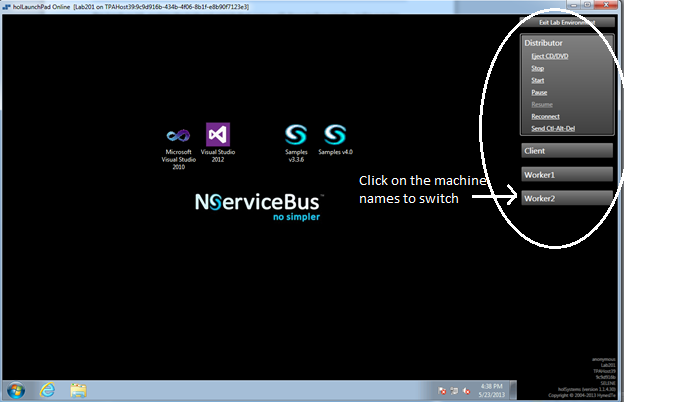
This hands on lab explains the ScaleOut sample that comes with NServiceBus samples. In this exercise, you will build the ScaleOut exercise step by step. You will also use Unobtrusive message conventions and define the message classes as POCOs as opposed to referencing NserviceBus in teh message library. When the need for scaling arises, you will see how to deploy an endpoint to multiple machines without changing code, by changing just the feature profile and minimal configuration.

Lab Machines

To see the ScaleOut in action, this lab is set up as four machines:

|  |  |
| --- | --- |
| Machine Name | Purpose |
| Distributor | This VM hosts the Retail.Orders.Handler endpoint, which runs either in the master profile or the distributor profile, which acts as the load balancer. |
| Client | This VM hosts the Retail.Orders.Client endpoint, which simulates the load to the distributor. |
| Worker1 | This VM hosts the same Retail.Orders.Handler endpoint running in the distributor machine, and instead runs it in the worker profile. This endpoint handles the load from the distributor. |
| Worker2 | This VM hosts the same Retail.Orders.Handler endpoint running in the distributor machine, and instead runs it in the worker profile. This endpoint handles the load from the distributor. |

To switch machines, click the buttons labeled “Client”, “Worker1”, or “Worker2” that are listed on the right top corner of the lab.



**Firewall settings:**

The following programs have been set as Allowed in the firewall configuration:

* All machines are allowed for File& Print Sharing, ICMP, Core Networking, MSMQ, Network Discovery, DTC and Visual Studio.
* All machines have DTC enabled and allow ports 5000-6000 and other settings configured as shown here:

<http://particular.net/articles/transactions-message-processing>

* The distributor machine has RavenDB port 8080 enabled.
* All of the VMs in this lab have unique MSMQ QMId in the registry and also a unique CID for the MSDTC in the registry, as described in the following articles:

<http://blogs.msdn.com/b/johnbreakwell/archive/2007/02/06/msmq-prefers-to-be-unique.aspx>

<http://www.wadewegner.com/2007/08/warning-the-cid-values-for-both-test-machines-are-the-same/>

**Estimated time to complete:**

60 mins.

Terms of Use

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by NServiceBus in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to NServiceBus. You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ONLAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

**FEEDBACK**. If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to NServiceBus, you give to NServiceBus, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a NServiceBus software or service that includes the feedback. You will not give feedback that is subject to a license that requires NServiceBus to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

NServiceBus Ltd. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB , INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. NServiceBus DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

Contents

[Exercise 1: Scale out using MSMQ with master/worker 6](#_Toc362424614)

[Task 1: Create a server endpoint and configure it to use MSMQ as transport 6](#_Toc362424615)

[Task 2: Create an endpoint to create load 8](#_Toc362424616)

[Task 3: Run the solution 11](#_Toc362424617)

[Task 4: Start the Retail.Orders.Handler endpoint to startup as master 14](#_Toc362424618)

[Task 5: Deploy Retail.Orders.Sender to the client machine 16](#_Toc362424619)

[Task 6: Deploy Retail.Orders.Handler to the Worker1 machine 18](#_Toc362424620)

[Task 7: Deploy Retail.Orders.Handler to the Worker2 machine 20](#_Toc362424621)

[Task 8: Simulate load and watch the processing 21](#_Toc362424622)

[Exercise 2: Switch to distributor/worker profiles 21](#_Toc362424623)

[Task 1: Run the Retail.Orders.Handler using the distributor profile 21](#_Toc362424624)

[Task 2: Simulate load and watch the processing 22](#_Toc362424625)

[Completed Lab Solution 23](#_Toc362424626)

[Running the completed solution 23](#_Toc362424627)

# Exercise 1: Scale out using MSMQ with master/worker

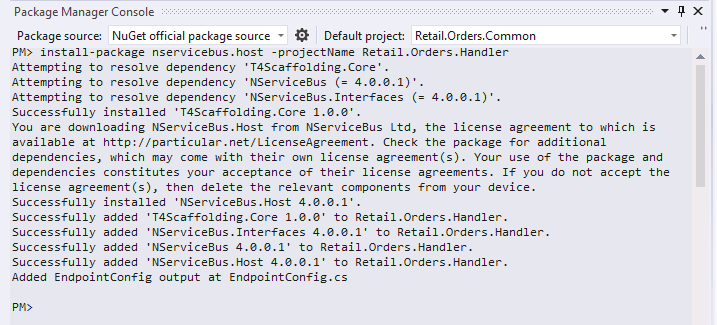
## Task 1: Create a server endpoint and configure it to use MSMQ as transport

1. Switch to the Distributor Machine (by clicking on Distributor on the right side of the lab window, where all the servers are listed) if the Distributor machine is not the currently active machine.
2. Open a new instance of Visual Studio 2012 and make sure to run it as an administrator.
3. Open the solution called Retail.Scaleout from C:\Hands on Labs\Exercises

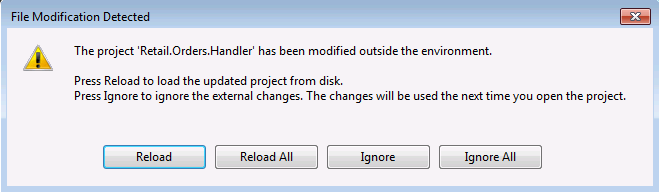
NOTE: The Retail.Orders.Messages project already has PlaceOrder command and OrderPlaced event messages defined as POCOs and the Retail.Orders.Common defines the unobtrusive conventions.

1. In the project Retail.Orders.Handler, add a project reference to the Retail.Orders.Messages project and Retail.Orders.Common.
2. In the Package Manager Console, type

**Install-package NServiceBus.Host –ProjectName Retail.Orders.Handler**



1. NServiceBus automatically adds the proper configuration for the endpoint. When prompted to reload the project, choose Reload All.



1. In your solution explorer is a new file called EndpointConfig.cs. NServiceBus configures the transport to MSMQ by default and configures a server endpoint. Change the configuration to mark this endpoint as a publisher, as shown:

namespace Retail.Orders.Handler

{

using NServiceBus;

public class EndpointConfig : IConfigureThisEndpoint, AsA\_Publisher

{

}

}

1. In file ProcessOrderCommandHandler.cs and implement the interface IhandleMessages<PlaceOrder> as shown:

namespace Retail.Orders.Handler

{

using System;

using NServiceBus;

using Retail.Orders.Messages.Commands;

using Retail.Orders.Messages.Events;

public class ProcessOrderCommandHandler : IHandleMessages<PlaceOrder>

{

public IBus Bus { get; set; }

public void Handle(PlaceOrder message)

{

Console.WriteLine("Received PlaceOrder command, order Id: {0}", message.OrderId);

// Publish the event

Bus.Publish<OrderPlaced>(m => m.OrderId = message.OrderId);

Console.WriteLine("Published OrderPlaced event for orderId: {0}", message.OrderId);

}

}

}

1. Compile your solution and make sure it builds as expected.

## Task 2: Create an endpoint to create load

1. In the project Retail.Orders.Sender add a project reference to the Retail.Orders.Messages and Retail.Orders.Common
2. In the Package Manager Console, type

**Install-package NServiceBus.Host –ProjectName Retail.Orders.Sender**

1. NServiceBus automatically adds the proper configuration necessary for the endpoint. When prompted to reload the project, choose Reload All.
2. In your solution explorer is a new file called EndpointConfig.cs, which looks like that below. NServiceBus configures the transport to MSMQ by default and configures a server endpoint. Leave the defaults.

namespace Retail.Orders.Sender

{

using NServiceBus;

public class EndpointConfig : IConfigureThisEndpoint, AsA\_Server

{

}

}

1. To send the PlaceOrder command to the Retail.Orders.Handler endpoint and subscribe to the OrderPlaced event, add the appropriate configuration in app.config.

In the UnicastBusConfig section, specify the event that is of interest to this subscriber, the endpoint (queue address) responsible for publishing this event, the command that needs to be sent, and the endpoint responsible for handling the command, as follows:

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<configuration>

<configSections>

<section name="MessageForwardingInCaseOfFaultConfig" type="NServiceBus.Config.MessageForwardingInCaseOfFaultConfig, NServiceBus.Core" />

<section name="UnicastBusConfig" type="NServiceBus.Config.UnicastBusConfig, NServiceBus.Core" />

</configSections>

<MessageForwardingInCaseOfFaultConfig ErrorQueue="error" />

<UnicastBusConfig ForwardReceivedMessagesTo="audit">

<MessageEndpointMappings>

<!--To register a specific type in an assembly -->

<add Assembly="Retail.Orders.Messages" Type="Retail.Orders.Messages.Events.OrderPlaced" Endpoint="Retail.Orders.Handler" />

<add Assembly="Retail.Orders.Messages" Type="Retail.Orders.Messages.Commands.PlaceOrder" Endpoint="Retail.Orders.Handler" />

</MessageEndpointMappings>

</UnicastBusConfig>

</configuration>

1. Once the appropriate configuration entries are added, the endpoint also needs to implement a message handler that will be invoked when the event is raised. In the file, OrderPlacedHandler.cs, implement the interface IHandleMessages<OrderPlaced>, and resolve the using statements.

namespace Retail.Orders.Sender

{

using System;

using NServiceBus;

using Retail.Orders.Messages.Events;

public class OrderPlacedHandler : IHandleMessages<OrderPlaced>

{

public void Handle(OrderPlaced message)

{

Console.WriteLine("Received Event OrderPlaced for orderId: {0}", message.OrderId);

}

}

}

1. In class ProcessOrderSender.cs and add code such that, every time a key is pressed, the endpoint sends ten PlaceOrder commands. This way, pressing enter multiple can simulate the necessary load.

namespace Retail.Orders.Sender

{

using System;

using NServiceBus;

using Retail.Orders.Messages.Commands;

using Retail.Orders.Messages.Events;

class ProcessOrderSender : IWantToRunWhenBusStartsAndStops

{

public IBus Bus { get; set; }

public void Start()

{

Console.WriteLine("Press 'Enter' to send 10 messages. To exit, Ctrl + C");

var orderNumber = 1;

while (Console.ReadLine() != null)

{

for (int index = 0; index < 10; index ++)

{

var orderId = string.Format("order{0}", orderNumber);

Bus.Send<PlaceOrder>(m => { m.OrderId = orderId; });

Console.WriteLine(string.Format("Sent PlacedOrder command with orderId: {0}.", orderId));

orderNumber++;

}

Console.WriteLine("Press 'Enter' to send another 10 messages. To exit, Ctrl + C");

}

}

public void Stop()

{

Bus.Unsubscribe<OrderPlaced>();

}

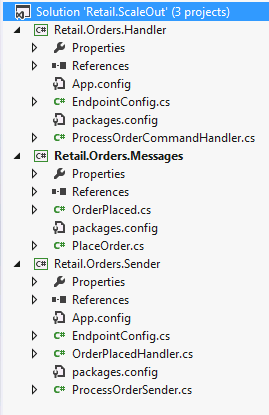
}

}

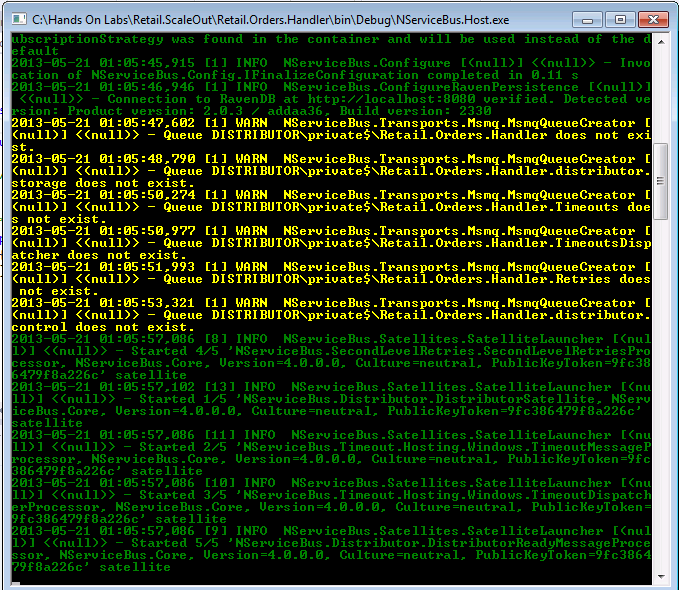
1. Compile your solution and make sure it builds as expected.

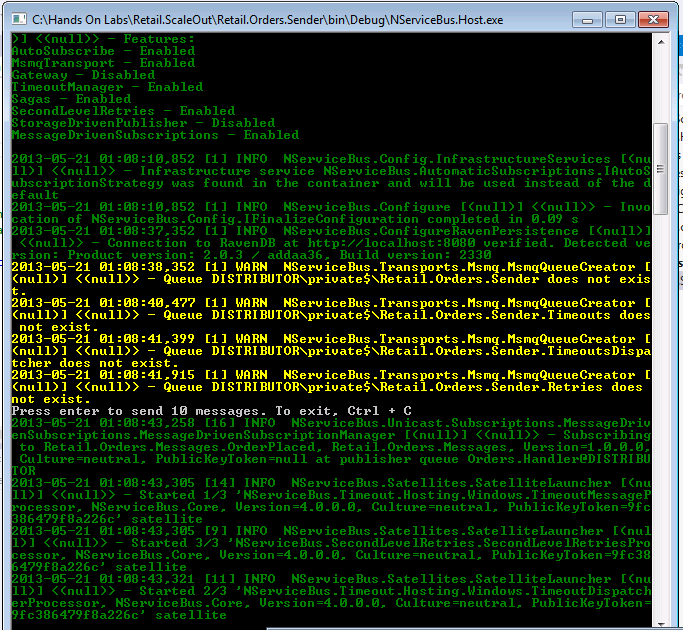
## Task 3: Run the solution

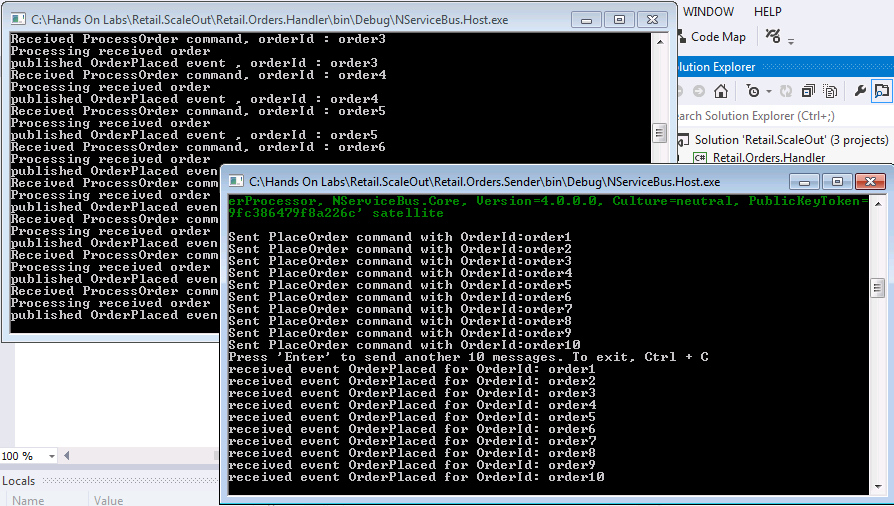
1. Your project solution should look like that below:



1. Start the Retail.Orders.Handler (Right click the project -> Debug ->Start new instance)

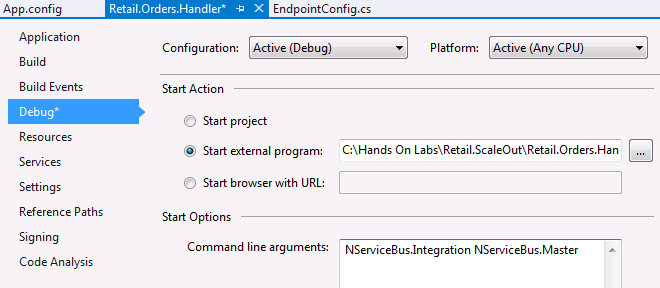


1. Start the Retail.Orders.Sender endpoint (Right click the project -> Debug ->Start new instance):
2. Press the Enter key in Retail.Orders.Sender in order to send ten PlaceOrder commands to the endpoint Retail.Orders.Handler. Ensure that the Retail.Orders.Handler receives all the commands and the sender endpoint receives all the published OrderPlaced events for the each of the commands that were sent.



## Task 4: Start the Retail.Orders.Handler endpoint to startup as master

1. Stop the endpoints.
2. Set up the command line argument to include NServiceBus.Integration and NServiceBus.Master as shown in the Project properties for Retail.Orders.Handler:



***About the NServiceBus.Master profile:*** When running under the master profile, when the endpoint starts up as a load balancer, this node also processes the messages that are received, in addition to delegating work to the other worker nodes.

***About the NServiceBus.Integration Profile*:** When running the endpoint from the command line, using the NServiceBus.Integration profile allows the queues to be created on startup, while still using RavenDB for subscription storage. Running the endpoint via the debugger in Visual Studio creates the queues even without the use of the NServiceBus.Integration profile. For production deployment, queues are created at install time, when using the /install switch. See the Installation section under

<http://particular.net/articles/the-nservicebus-host#installation>.

1. Modify app.config and ensure that the configuration is set up appropriately to scale up by setting the MaximumConcurrency level defined in TransportConfig section.

To quickly add the TransportConfig section to app.config, go to Package Manager Console and type: **add-NServiceBusTransportConfig –projectName Retail.Orders.Handler** and when prompted to reload, click Yes.

Adjust the MaximumConcurrencyLevel in the TransportConfig section, as shown to process five messages at a time:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="MessageForwardingInCaseOfFaultConfig" type="NServiceBus.Config.MessageForwardingInCaseOfFaultConfig, NServiceBus.Core" />

<section name="TransportConfig" type="NServiceBus.Config.TransportConfig, NServiceBus.Core"/>

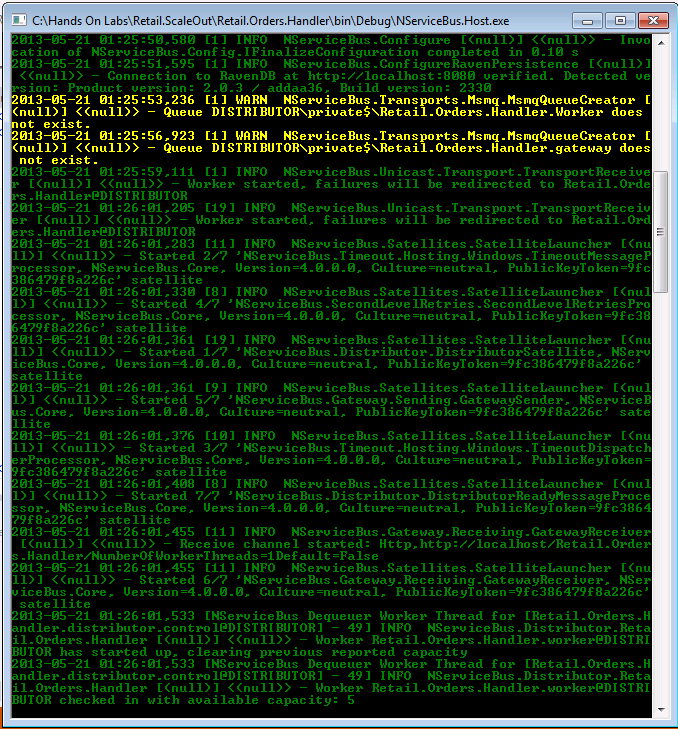
</configSections>

<MessageForwardingInCaseOfFaultConfig ErrorQueue="error"/>

<TransportConfig MaxRetries="2" MaximumConcurrencyLevel="5" MaximumMessageThroughputPerSecond="0"/>

</configuration>

1. Start the Retail.Orders.Handler endpoint. This should now run using the Master profile:



1. The binaries will be available to the other virtual machines via the File Share [\\Distributor\Exercises](file:///\\Distributor\Exercises), in order to facilitate deployment of the code to the other machines.

## Task 5: Deploy Retail.Orders.Sender to the client machine

1. Switch to the client machine by clicking Client on the right side of the lab window, where the servers are listed.
2. On this machine, copy the binaries from

[\\Distributor\Exercises\Retail.Orders.Sender\bin\debug](file:///\\Distributor\Exercises\Retail.Orders.Sender\bin\debug) to C:\Hands On Labs\Deployment.

1. Change the Retail.Orders.Sender.dll.config in C:\Hands On Labs\Deployment, so that the messages are sent to the master node, i.e. the Distributor:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="Logging" type="NServiceBus.Config.Logging, NServiceBus.Core" />

<section name="UnicastBusConfig" type="NServiceBus.Config.UnicastBusConfig, NServiceBus.Core" />

<section name="MessageForwardingInCaseOfFaultConfig" type="NServiceBus.Config.MessageForwardingInCaseOfFaultConfig, NServiceBus.Core" />

</configSections>

<MessageForwardingInCaseOfFaultConfig ErrorQueue="error"/>

<Logging Threshold="INFO" />

<UnicastBusConfig ForwardReceivedMessagesTo="audit">

<MessageEndpointMappings>

<!--To register a specific type in an assembly -->

<add Assembly="Retail.Orders.Messages" Type="Retail.Orders.Messages.Events.OrderPlaced" Endpoint="Retail.Orders.Handler@Distributor" />

<add Assembly="Retail.Orders.Messages" Type="Retail.Orders.Messages.Commands.PlaceOrder" Endpoint="Retail.Orders.Handler@Distributor" />

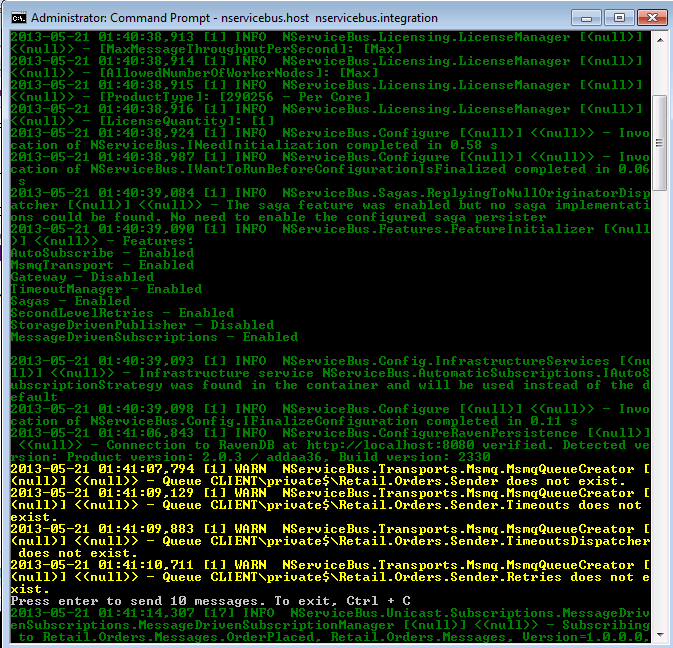
</MessageEndpointMappings>

</UnicastBusConfig>

</configuration>

1. At an administrative command prompt, change the directory to the correct location (i.e., C:\Hands on Labs\Deployment). Start the sender in the Integration profile as follows:

**NServiceBus.Host NServiceBus.Integration**



## Task 6: Deploy Retail.Orders.Handler to the Worker1 machine

1. Switch to the **Worker1** machine by clicking Worker1 on the right side of the lab window, where the servers are listed.
2. On this machine, copy the binaries from \\Distributor\Exercises\Retail.Orders.Handler\bin\debug to C:\Hands on Labs\Deployment.
3. Modify the Retail.Orders.Handler.dll.config in this location and ensure that the configuration is set up appropriately in app.config to:

* Ensure that the master node configuration is properly specified, as shown below.
* Specify the distributor control address and the application queue located at the master node in UnicastBusConfig.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="MessageForwardingInCaseOfFaultConfig" type="NServiceBus.Config.MessageForwardingInCaseOfFaultConfig, NServiceBus.Core" />

<section name="TransportConfig" type="NServiceBus.Config.TransportConfig, NServiceBus.Core"/>

<section name="UnicastBusConfig" type="NServiceBus.Config.UnicastBusConfig, NServiceBus.Core" />

<section name="MasterNodeConfig" type="NServiceBus.Config.MasterNodeConfig, NServiceBus.Core" />

</configSections>

<MasterNodeConfig Node="Distributor"/>

<MessageForwardingInCaseOfFaultConfig ErrorQueue="error"/>

<TransportConfig MaximumConcurrencyLevel="5" MaxRetries="2"/>

<UnicastBusConfig ForwardReceivedMessagesTo="audit" DistributorControlAddress="retail.orders.handler.distributor.control@Distributor"

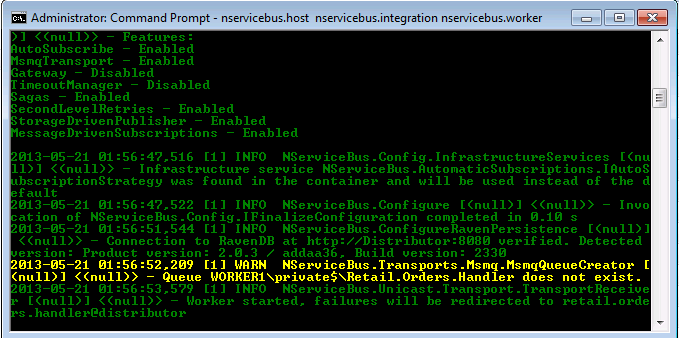
DistributorDataAddress="retail.orders.handler@Distributor">

</UnicastBusConfig>

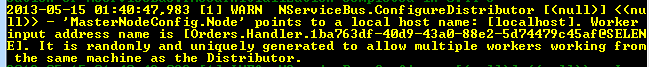
</configuration>

1. At an administrative command prompt, change the directory to C:\Hands on Labs\Deployment Start the endpoint in the worker mode by typing

**NServiceBus.Host NServiceBus.Integration NServiceBus.Worker**



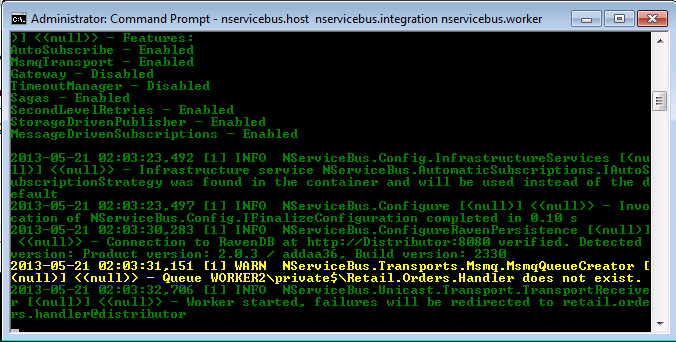
***About the NServiceBus.Worker Profile***: Running the endpoint in the Worker profile sets up the endpoint as a worker that communicates with the distributor to receive the load when the endpoint starts. If the worker is run on the same machine as the master node, then an automatically generated unique name for the worker is created (at install time, when using the NServiceBus.Production profile; or at runtime, when using the NServiceBus.Integration profile):



## Task 7: Deploy Retail.Orders.Handler to the Worker2 machine

1. Switch to the Worker2 machine by clicking Worker2 on the right side of the lab window, where the servers are listed.
2. Copy the binaries from [\\Worker1\Deployment](file:///\\Worker1\Deployment) to C:\Hands on Labs\Deployment
3. At an administrative command prompt, change the directory to C:\Hands on Labs\Deployment and start the endpoint in the worker mode by typing:

**NServiceBus.Host NServiceBus.Integration NServiceBus.Worker**



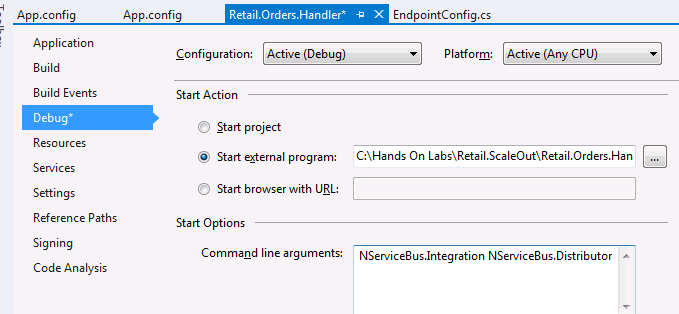
## Task 8: Simulate load and watch the processing

1. Switch to the client machine by clicking Client on the right side, where the servers are listed.
2. Press the Enter key a few times to simulate a load of messages that are sent from the Retail.Orders.Sender running on the client machine.
3. Switch to the Distributor machine where the Retail.Orders.Handler host is running using the NServiceBus.Master profile and watch the endpoint process messages in addition to distributing the load.
4. Switch to the Worker1 machine and watch the Retail.Orders.Handler.
5. Switch to the Worker2 machine and watch the Retail.Orders.Handler.
6. Switch to the client machine and wait to receive the responses for all messages sent.

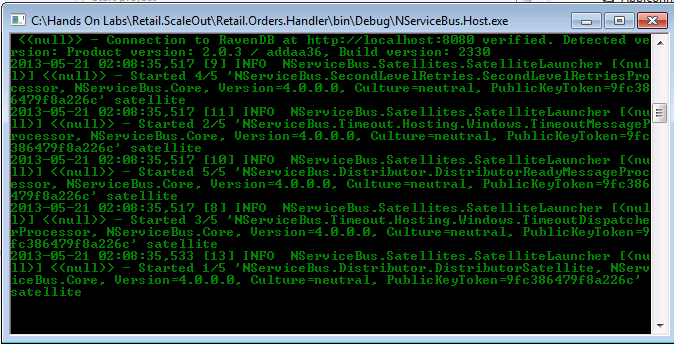
# Exercise 2: Switch to distributor/worker profiles

## Task 1: Run the Retail.Orders.Handler using the distributor profile

1. Switch to the distributor machine by clicking Distributor on the right side where the servers are listed.
2. To stop the endpoint, press Ctrl + C, or stop debugging on the VS2012 IDE.
3. Add the distributor profile instead of the master profile as the command line arguments in Project Properties for Retail.Orders.Handler, as shown:



***About the NServiceBus.Distributor profile*:**  When running in this profile, NServiceBus starts the endpoint in the distributor mode, but does not enlist itself as a worker.



## Task 2: Simulate load and watch the processing

1. Switch to the client machine by clicking Client on the right side where the servers are listed.
2. Press the Enter key a few times to simulate a load.
3. Switch to the distributor machine and watch the Retail.Orders.Handler. Note that it does not handle any messages; instead, it acts as a pure load balancer.
4. Switch to the Worker1 machine and watch the Retail.Orders.Handler.
5. Switch to the Worker2 machine and watch the Retail.Orders.Handler.
6. Switch to the client machine and wait to receive the responses for the PlaceOrder command messages that were sent.

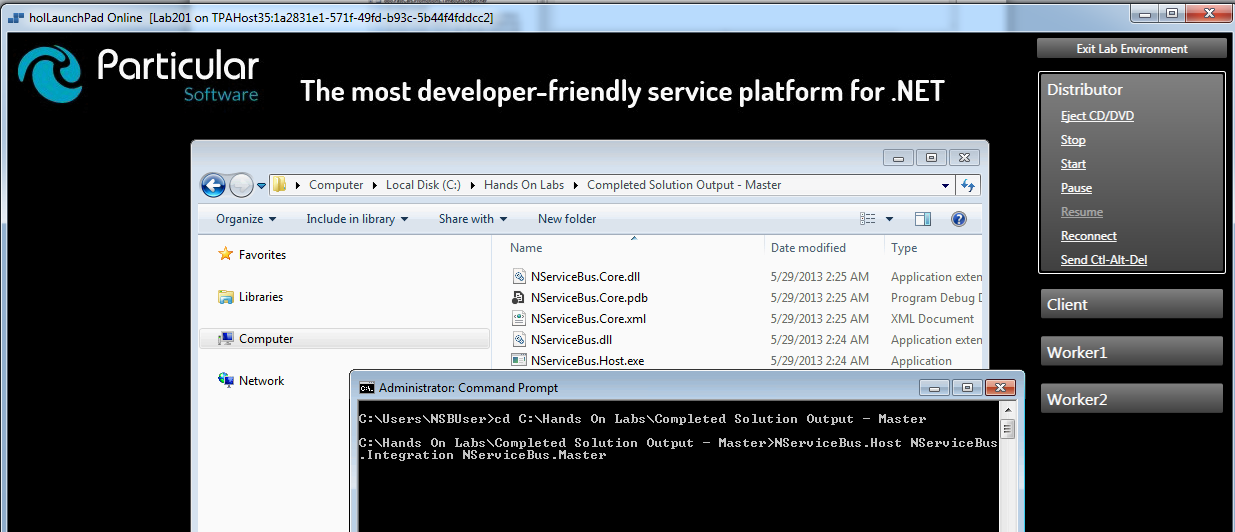
# Completed Lab Solution

For your convenience, the complete source code for this solution is available on the Distributor machine at C:\Hands on Labs\Completed Solution Source Code.

## Running the completed solution

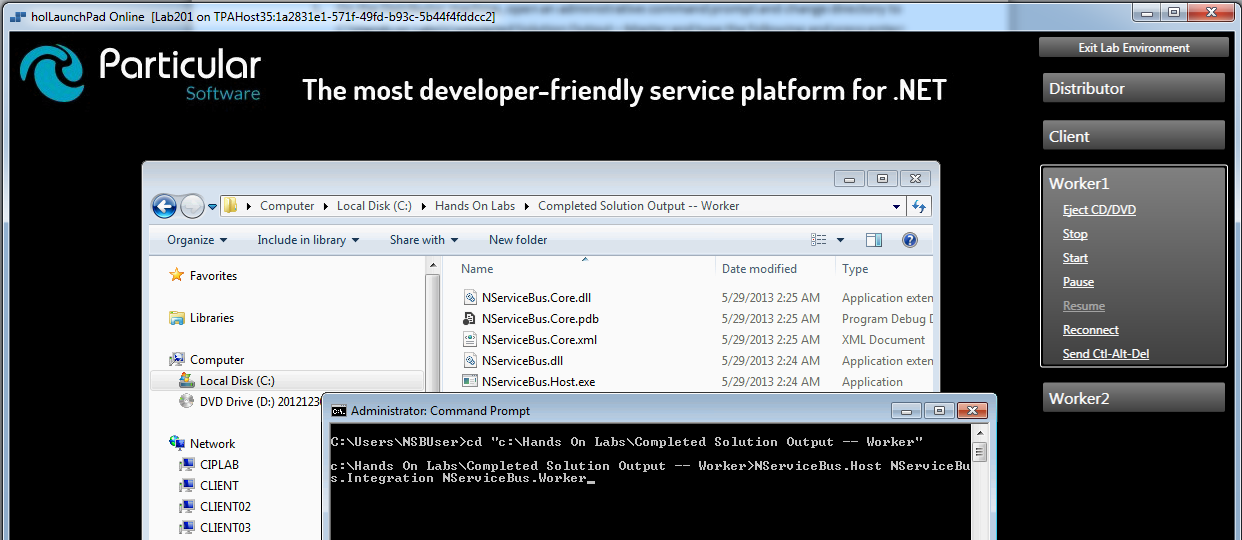
1. On the Distributor machine, open an administrative command prompt and change directory to C:\Hands on Labs\Completed Solution Output – Master and type:

**NServiceBus.Host NServiceBus.Integration NServiceBus.Master**



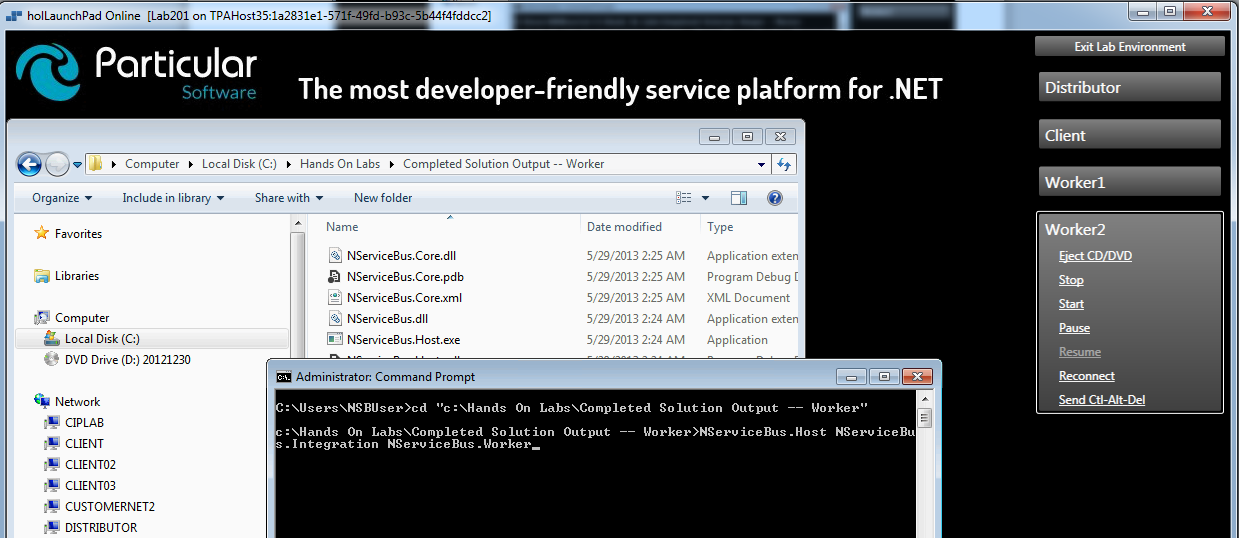
1. Switch to the Worker1 machine and start the worker node. To do this, open an administrative command prompt and change directory to C:\Hands on Labs\Completed Solution Output -- Worker and type the following and press enter.

**NServiceBus.Host NServiceBus.Integration NServiceBus.Worker**



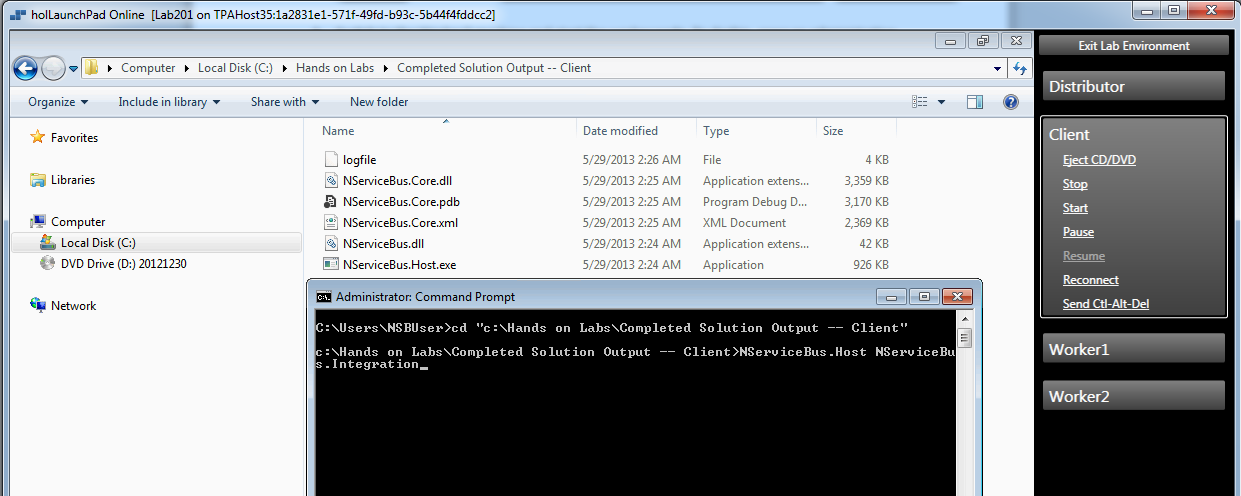
1. Switch to the Worker2 machine and start the worker node. To do this, open an administrative command prompt and change directory to C:\Hands on Labs\Completed Solution Output -- Worker and type:

**NServiceBus.Host NServiceBus.Integration NServiceBus.Worker**



1. Switch to the Client machine and start the sender application. To do this, open an administrative command prompt and change directory to C:\Hands on Labs\Completed Solution Output -- Client and type:

**NServiceBus.Host NServiceBus.Integration**



1. Press the Enter key a few times to generate load and switch to each of the machines and watch the master node distribute work to the workers.