APM for .NET

AppDynamics Pro Documentation

Version 4.0.x

APM for .NET

Once you instrument your .NET application environment with the .NET Agent, you can start monitoring .NET application performance. For general information on configuring AppDynamics deployment, see the topics under AppDynamics Essentials.

This section describes considerations specific for monitoring .NET applications. It also describes how to fine tune the configuration for specific .NET frameworks and technologies. These include, for example, monitoring and configuration topics for your IIS applications, Windows services, and standalone applications.

Get a PDF version of the .NET monitoring documentation.

End-to-end APM for .NET

.NET Supported Environments
Install a Controller or get a SaaS Controller
Instrument .NET Applications
Monitor CLRs
Monitor Windows Hardware Resources

What's New in .NET Monitoring in 4.0?

- Development mode
- Percentile metrics
- Business transaction lock down
- See more in the release notes

Learn by watching

More videos...

Configure AppDynamics for .NET

- Configure Business Transaction Detection for .NET
- Getter Chains in .NET Configurations
- Configure Service Endpoints for .NET
- Configure Backend Detection for .NET

Configure Business Transaction Detection for .NET

On this page:

- .NET Entry Points

Related pages:

- Business Transaction Entry Points
- Configure Business Transaction Detection
- Organize Traffic as Business Transactions
- Import and Export Transaction Detection Configuration

This topic introduces .NET entry points and the way that they are used in business transaction detection. If the auto-discovered entry points don't include all your critical business transactions, use custom match rules to customize entry point discovery.

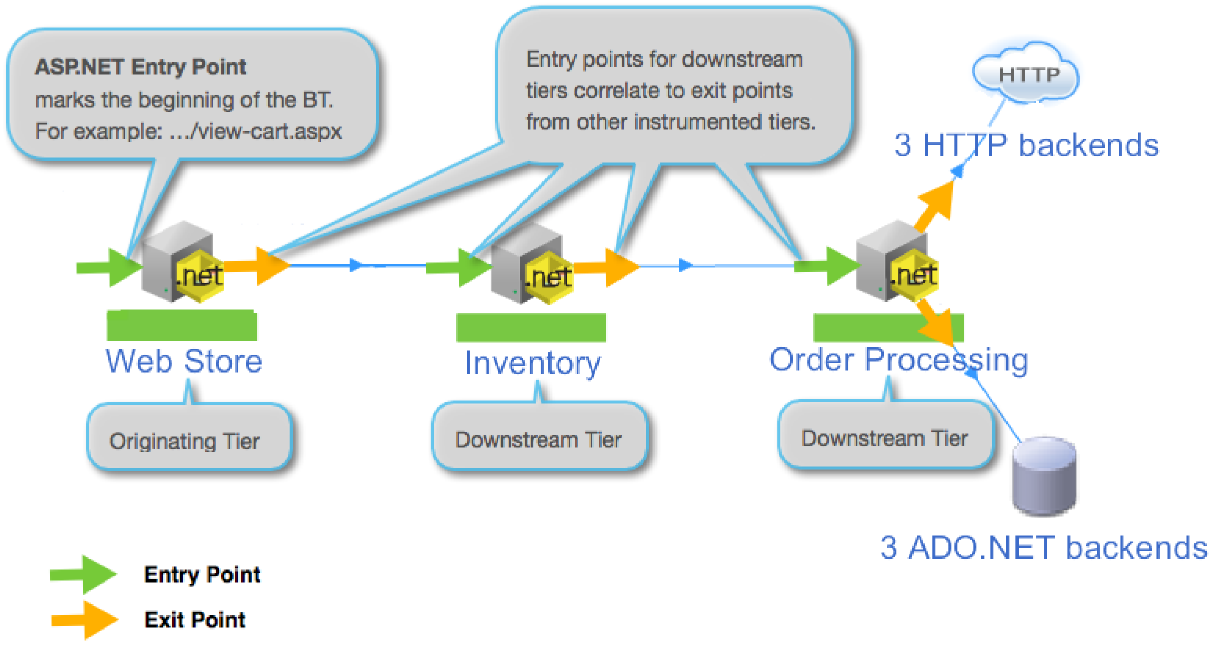
.NET Entry Points

AppDynamics detects entry points in the following places:

- On **originating tiers**, the method or operation that marks the beginning of a business transaction is an entry point. In most cases, this type of entry point maps to a user request or action such as "View/Cart". Entry points on originating tiers define the business transaction name.
- On **downstream tiers**, entry points correlate to incoming http calls, web service requests, and other communications from instrumented tiers.

The .NET Agent (agent) automatically detects entry points for the frameworks listed as automatically discovered business transactions. See Supported Environments and Versions (.NET).

If the agent detects an entry point on the originating tier, it registers a business transaction with the Controller. The Controller displays the business transaction in the Business Transactions list. If an entry point correlates to an upstream exit point, such as an exiting service call, the agent includes it as part of the existing business transaction.



If you need to monitor transactions not automatically discovered by the App Agent for .NET, you can customize entry point detection. See Organize Traffic as Business Transactions to learn how to plan your business transactions. After you have identified the critical business transactions for your application, create custom match rules for them to enable transaction detection.

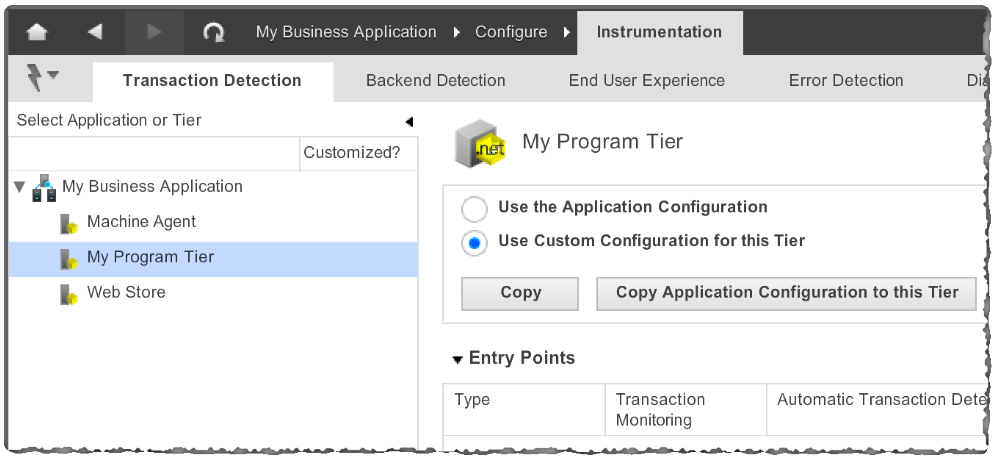
To create custom match rules for .NET entry points:

1. Click **Configure > Instrumentation > Transaction Detection**.

2. Click the **.NET - Transaction Detection** tab.

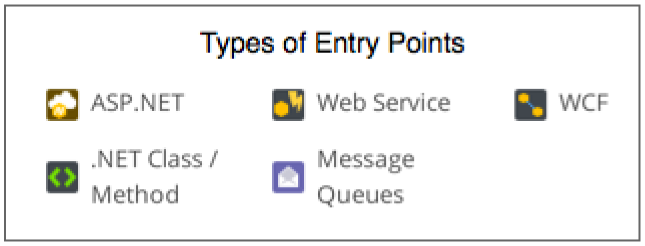
3. From the **Select Application or Tier** list at the left, click either:

- an application, to configure transaction detection for all tiers in a business application.
- a tier, to configure transaction detection at the tier level. At the tier level click **Use Custom Configuration for this Tier**. AppDynamics copies the application configuration to the tier level so that you can modify it for the tier.



4. Use the **Custom Match Rules** pane to add and remove business transaction match rules. For details on types of .NET entry points and how to setup custom match rules, see:

- ASP.NET Entry Points
- Message Queue Entry Points
- POCO (.NET Class/Method) Entry Points
- WCF Entry Points
- Web Service Entry Points



POCO Entry Points

On this page:

- Defining a POCO Entry Point
- Discovery of POCO Transactions

Related pages:

- Monitor Background Tasks
- Configure Business Transaction Detection
- Business Transaction Entry Points

Some applications use frameworks that the .NET Agent doesn't automatically detect. This is frequently the case with Windows services and standalone applications. AppDynamics lets you specify entry points using custom match rules for Plain Old CLR Objects (POCOs). Once you've defined POCOs, we measure performance data for POCO transactions the same as for other transactions.

Define the custom match rule on the .NET class/method that is the most appropriate entry point for the business transaction (BT). Someone who is familiar with your application code should help make this determination.

Defining a POCO Entry Point

On an originating tier, a POCO entry point is the method that starts the BT. If the POCO entry point is on a downstream tier, it may correlate to an upstream exit point. When defining a POCO entry point, it is important to choose a method that begins and ends every time the BT executes. For more on entry points, see Monitor Business Transactions.

Good candidates for POCO entry points include the following:

- A timer in a Windows service that executes a database call to check for new records to process. For example, an order processing system that periodically checks for new orders in the system.
- A loop in a standalone application that batch processes records via a web service call. For example, an expense reporting system that loops through approved expenses to submit them for reimbursement.
- A method in a web application that executes every time a client connects. For example, consider the method execution sequence:

```
using System.Net;
using System.Net.Sockets;
using System.Threading.Tasks;

namespace MyService
{
class MyProgram
{
    static void Main(string[] args)
    {
        TcpListener myList = new TcpListener(IPAddress.Parse("127.0.0.1"),
8000);
        using (Socket s = myList.AcceptSocket())
            Task.Factory.StartNew(DoWork, s);
    }
    static void DoWork<Socket>(Socket s)
    {
        // ...
    }
}
}
```

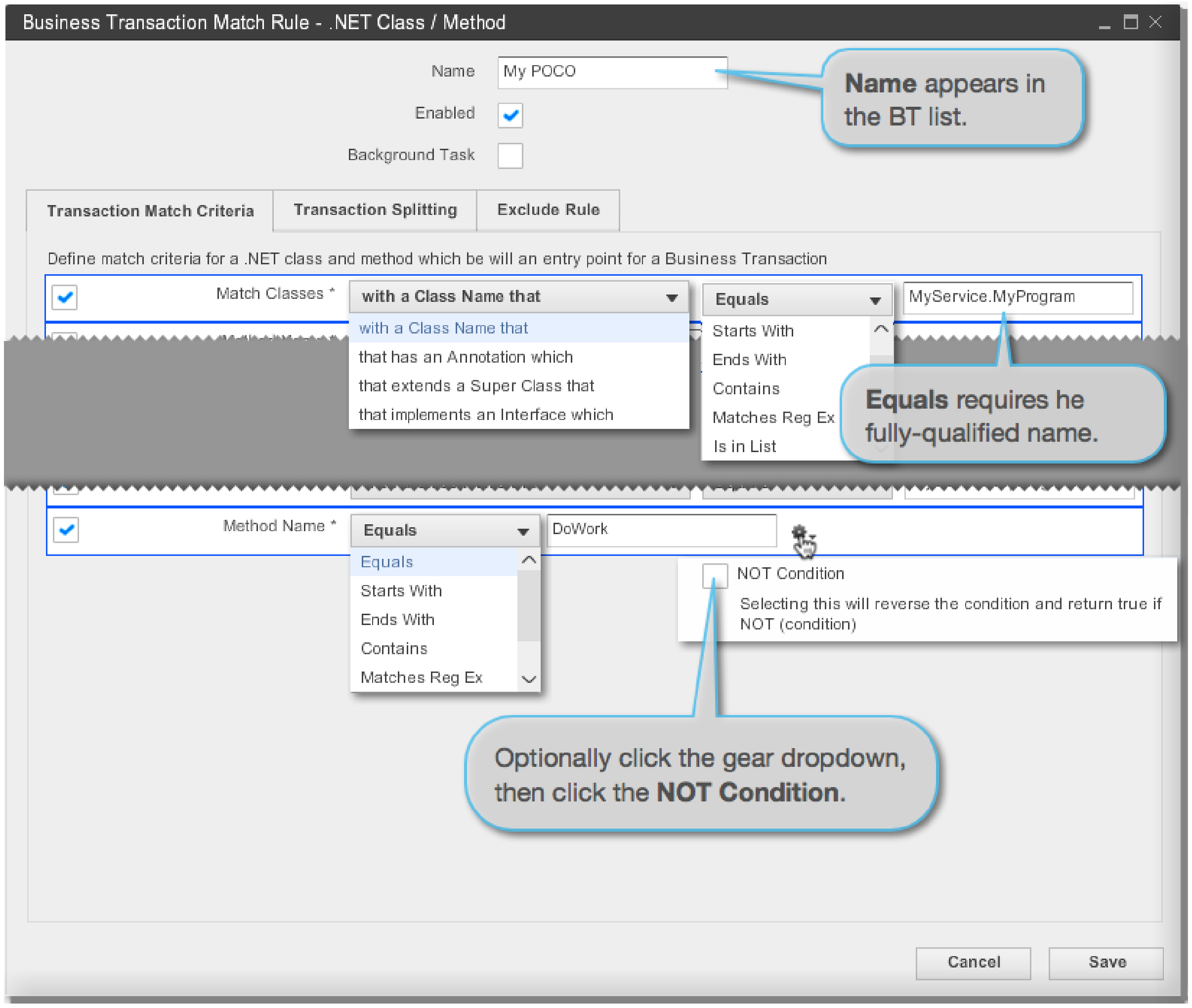
The `AcceptSocket()` method is the blocking method that accepts a job and invokes it. The `MyP` `rogram.DoWork()` method is the unit of work because it executes every time a client calls the business transaction and it finishes at the same time as the business transaction. This makes `DoWork()` a good POCO entry point.

Discovery of POCO Transactions

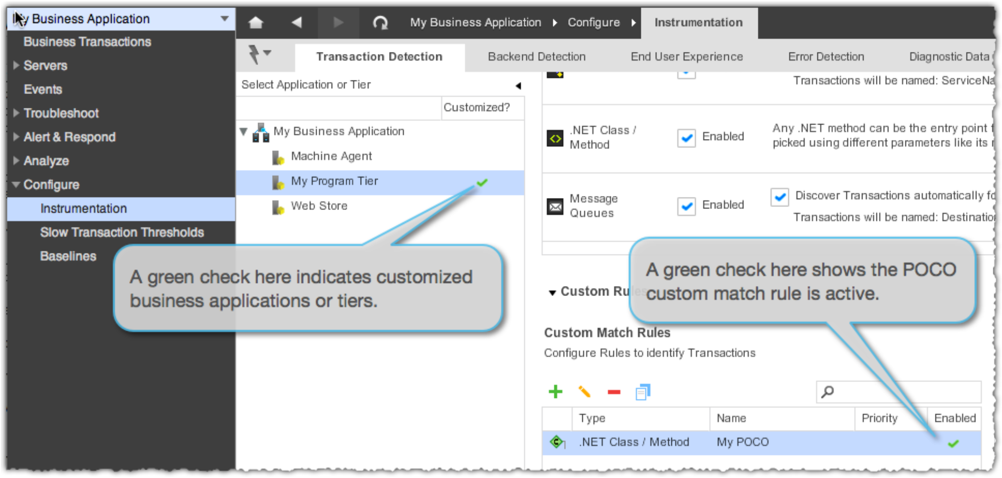
By default, once you configure a custom match rule for a POCO entry point, the .NET Agent detects and displays it in the Business Transactions list. AppDynamics names the BT for the name of the custom match rule. For more information, see Configure Business Transaction Detection.

To set up a POCO entry point, define a custom match rule for a .NET Class/Method. For steps to access the **Custom Match Rules** pane, see To create custom match rules for .NET entry points.

1. In the **Custom Match Rules** pane, click the plus symbol (**+**) to add an entry point.
2. Click **.NET Class/Method** in the dropdown list, then click **Next**.
3. Name the **New Business Transaction Match Rule**.
    - AppDynamics uses the rule **Name** to name the BT.
    - The Controller enables the rule by default. Disable it later if needed.
    - The POCO is a foreground task, to configure it as a background task, see POCO Transaction as a Background Task.
4. In the **Transaction Match Criteria** tab, specify the **Match Classes** criteria.
5. Specify the **Method Name** match criteria.

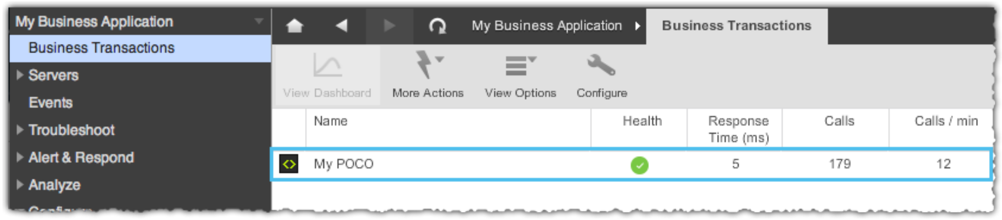

Business Transaction Match Rule - .NET Class / Method
Name: My POCO
Enabled: ✓
Background Task
Name appears in the BT list.
Transaction Match Criteria
Transaction Splitting
Exclude Rule
Define match criteria for a .NET class and method which be will an entry point for a Business Transaction
Match Classes *: with a Class Name that
with a Class Name that
that has an Annotation which
that extends a Super Class that
that implements an Interface which
Equals
Starts With
Ends With
Contains
Matches Reg Ex
Is in List
MyService.MyProgram
Equals requires he fully-qualified name.
Method Name *: Equals
Equals
Starts With
Ends With
Contains
Matches Reg Ex
DoWork
NOT Condition
Selecting this will reverse the condition and return true if NOT (condition)
Optionally click the gear dropdown, then click the NOT Condition.
Cancel
Save

6. Click **Create Custom Match Rule**. If you are configuring at the application level, click **Config ure all Tiers to use this Configuration**.
7. Click **OK** to the notification message **Instrumentation changes require restart**.
   After you save the rule, it appears in the **Custom Match Rule** list. The business application or tier you customized displays a green check in the **Select Application or Tier** pane.


My Business Application
Business Transactions
Servers
Events
Troubleshoot
Alert & Respond
Analyze
Configure
Instrumentation
Slow Transaction Thresholds
Baselines
My Business Application › Configure › Instrumentation
Transaction Detection
Backend Detection
End User Experience
Error Detection
Diagnostic Data
Select Application or Tier
Customized?
My Business Application
Machine Agent
My Program Tier ✓
Web Store
Transactions will be named: ServiceNa
.NET Class / Method
Enabled
Any .NET method can be the entry point picked using different parameters like its
Message Queues
Enabled
Discover Transactions automatically fo
Transactions will be named: Destinatio
Custom Ru
Custom Match Rules
Configure Rules to identify Transactions
Type
Name
Priority
Enabled
.NET Class / Method
My POCO
✓
A green check here indicates customized business applications or tiers.
A green check here shows the POCO custom match rule is active.

8. Wait one minute and restart the CLR/application.

The next time the POCO method executes, the agent detects it and registers the entry point. If the entry point is on an originating tier, the Controller displays it as a business transaction in the Business Transactions List.



Name	Health	Response Time (ms)	Calls	Calls / min
My POCO	✓	5	179	12

The agent identifies all the qualifying transactions using the custom match rule. In some situations you may need to further refine the discovery rules. Use the splitting options on the **T ransaction Splitting** tab.

If you create an application-level POCO that applies to a tier that has not yet registered with the Controller, you may need to restart the application after the tier registers in order to see the business transaction.

POCO Transaction as a Background Task

Click the **Background Task** check box in the **Business Transaction Match Rule** window to indicate that a POCO transaction runs in the background.

When a request runs as a background task, AppDynamics reports only Business Transaction metrics for the request. It does not aggregate response time and call metrics at the tier and application levels for background tasks. This prevents background tasks from distorting baselines for the business application. Also, you can set a separate set of thresholds for background tasks. See Monitor Background Tasks.

Learn More

- Monitor Background Tasks
- Configure Business Transaction Detection

ASP.NET Entry Points

On this page:

- Default Automatic Naming for ASP.NET Transactions
- Identify Transactions Using URI Segments
- Identify Transactions Using Headers, Cookies, and Other Parts of HTTP Requests
- Custom Match Rules for ASP.NET Transactions

Related pages:

- Configure Business Transaction Detection for .NET
- Business Transaction Entry Points
- Configure Business Transaction Detection

AppDynamics automatically detects entry points for client requests to ASP.NET applications. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. In most cases, this type of entry point maps to a user request or action such as "Cart/Checkout". AppDynamics allows you to configure transaction naming based upon the ASP.NET request.

Default Automatic Naming for ASP.NET Transactions

By default, the AppDynamics auto-detection naming scheme identifies all ASP.NET transactions using the first two segments of the URI.

For example, the following URI represents the checkout operation in an online store:

```
http://example.com/Cart/Checkout
```

AppDynamics automatically names the transaction:

```
/Cart/Checkout
```

For another example, the following URI represents a funds transfer operation in an online bank:

```
http://webbank.mybank.com/Account/Transferfunds/NorthernCalifornia
```

AppDynamics automatically names the transaction:

```
/Account/Transferfunds
```

Customize the Automatic Naming Scheme

AppDynamics lets you customize the auto-detected naming scheme to best fit your environment:

- Identify transactions using URI segments
- Identify transactions using headers, cookies, and other parts of HTTP requests

To modify automatic naming

1. Click **Configure > Instrumentation > Transaction Detection**.

2. Click the **.NET - Transaction Detection** tab.

3. From the **Select Application or Tier** list at the left, click either:
    - an application to configure transaction detection for all tiers in a business application.
    - a tier. At the tier level click **Use Custom Configuration for this Tier**. AppDynamics copies the application configuration to the tier level so that you can modify it for the tier.

4. If necessary, click **Enabled** under Transaction Monitoring and **Discover Transactions automatically for ASP.NET requests**.
   You can configure naming with Discover Transactions automatically for ASP.NET requests disabled, but the agent doesn't discover ASP.NET transactions.

5. Click **Configure Naming** for the ASP.NET type in the in the Entry Points panel.

6. Change the naming scheme in the ASP.NET Transaction Naming Configuration window and click **Save**.

The following sections provide examples to help you decide how to configure the naming scheme.

Identify Transactions Using URI Segments

AppDynamics offers the following options to automatically name ASP.NET transactions based upon the URI:

- Use all, first, or last URI segments
- Use specific URI segments

To name transactions using all, first, or last URI segments

Consider the following URL that represents the checkout operation in an online store:

```
http://example.com/Web/Store/Checkout
```

The first two segments of the URI don't provide a significant name for the business transaction:

```
/Web/Store
```

Identify a more meaningful name using one of the following options:

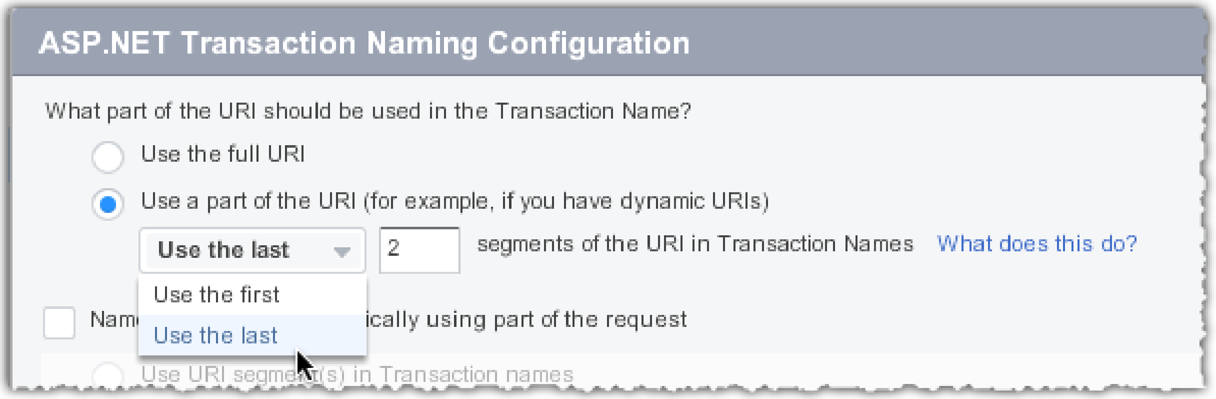
- Click **Use the full URI** to identify the transaction by all URI segments. For example:

  ```
  /Web/Store/Checkout
  ```

- Click **Use the first** or **Use the last** n segments to use two contiguous segments at the beginning or end of the URI, where n is the number of segments.

  For example, to identify the checkout transaction using the last two segments of the URI:

  ```
  /Store/Checkout
  ```



ASP.NET Transaction Naming Configuration

What part of the URI should be used in the Transaction Name?

( ) Use the full URI

(●) Use a part of the URI (for example, if you have dynamic URIs)

Use the last ▾ 2 segments of the URI in Transaction Names What does this do?

Use the first
Use the last
Nam
ically using part of the request
Use URI segm t(s) in Transaction names

- If you need more flexibility, such as using non-contiguous segments in the name, click **Name Transactions dynamically using part of the requests** and specify the segments with the **Use URI segments in Transaction names** option.

To use specific URI segments in transaction names

You can choose specific URI segments to use in the transaction name. This enables you to skip URI segments or use non-contiguous segments in the naming scheme.

1. Click **Use a part of the URI**.

2. Enter the number of first or last segments to use.

3. Click **Name Transactions dynamically using part of the request.**

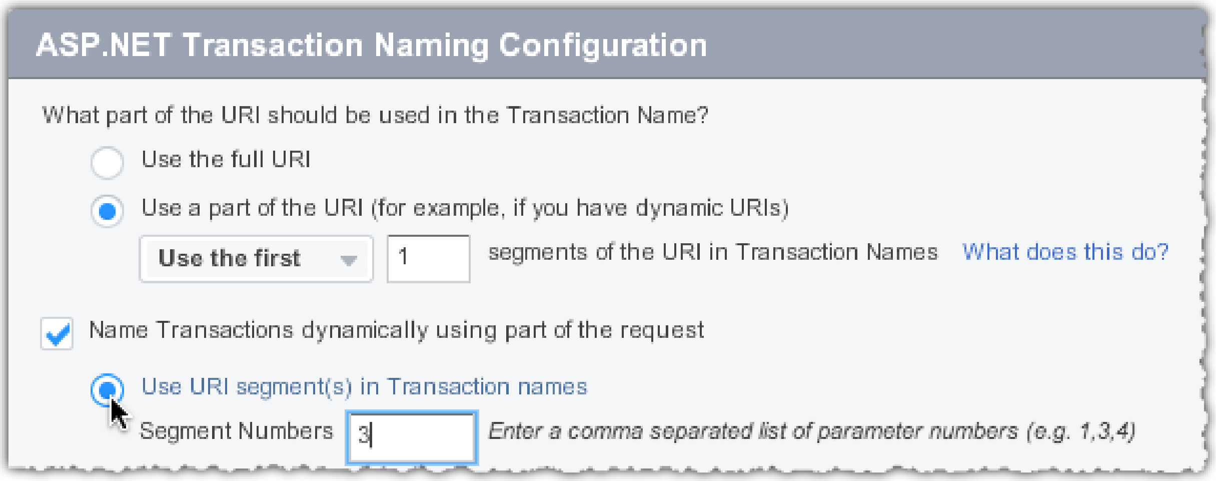
4. Click **Use URI segment(s) in Transaction names**.

5. Enter the segment numbers separated by commas.

   For example the following URL represents the checkout transaction requested by a customer with ID 1234:

   ```
   http://example.com/Store/cust1234/Checkout
   ```

   The checkout transaction is the same regardless of the customer, so it makes sense to name the transaction based upon the first and third segments of the URI.



ASP.NET Transaction Naming Configuration

What part of the URI should be used in the Transaction Name?

- Use the full URI
- [✓ Use a part of the URI (for example, if you have dynamic URIs)]

Use the first | 1 | segments of the URI in Transaction Names | What does this do?

[✓] Name Transactions dynamically using part of the request

[✓ Use URI segment(s) in Transaction names]

Segment Numbers | 3 | *Enter a comma separated list of parameter numbers (e.g. 1,3,4)*

AppDynamics names the transaction:

```
/Store/Checkout
```

Identify Transactions Using Headers, Cookies, and Other Parts of HTTP Requests

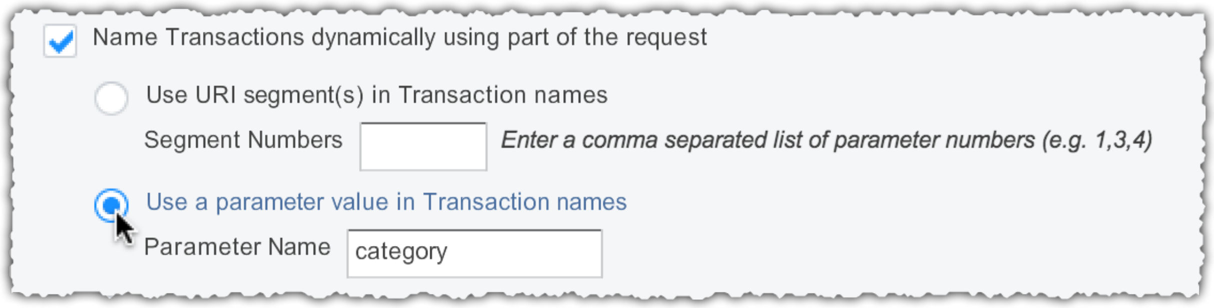
You can also name ASP.NET transactions using parameters, headers, cookies, and other parts of HTTP requests.

To identify all your ASP.NET transactions using particular parts of the HTTP request, use the Name **Transactions dynamically using part of the request** option.

Carefully consider your naming configuration choices. If you use a value such as the request originating address and you have many clients accessing your application, you may see the All Other Traffic Business Transaction.

To use HTTP parameter values in transaction names

1. Set the URI identification option.
2. Click **Use a parameter value in Transaction names** and enter the **Parameter Name**. For example, consider the following URL: http://example.com/Store/Inventory?category=electronics



[✓] Name Transactions dynamically using part of the request

Use URI segment(s) in Transaction names

Segment Numbers — *Enter a comma separated list of parameter numbers (e.g. 1,3,4)*

[● Use a parameter value in Transaction names]

Parameter Name: category

AppDynamics names the transaction to include the category parameter value: `/Store/Inventory`.electronics

To use a header value in transaction names

1. Set the URI identification option.
2. Click **Use header value in transaction names** and enter a **Header Name.**
   For example, consider a site that uses the custom header "Version", AppDynamics names transactions with the header value as follows: `/Store/Inventory.v2.5`

Use a cookie value in transaction names

1. Set the URI identification option.
2. Click **Use a cookie value in Transaction names** and enter the **Cookie Name**.
   For example, a website tracks a user's loyalty status in a cookie. Set the Cookie Name to "loyalty". AppDynamics names transactions for the loyalty cookie value: `/Store/Inventory.Status=Gold`

Use a session attribute value in transaction names

1. Set the URI identification option.
2. Click **Use a session attribute in Transaction names** and enter the **Session Attribute Key**.
   For example, a website stores a customer's region in the session property. Set the Session Attribute name to "region". AppDynamics names transactions for the region session attribute value: `/Store/Inventory.NorthAmerica`

Use the request method in Transaction names

1. Set the URI identification option.
2. Click **Use the request method (GET/POST/PUT) in Transaction names**.
   AppDynamics names transactions for the request method. For example: `/Store/Inventory.GET`

Use the request host in Transaction names

1. Set the URI identification option.
2. Click **Use the request host in Transaction names**.
   AppDynamics names transactions for the ip address of the request host. For example: `/Store/Inventory.192.0.2.0`

Use the request originating address in Transaction names

1. Set the URI identification option.
2. Click **Use the request originating address in Transaction names**.
   AppDynamics names transactions for the ip address of the request client. For example: `/Store/Inventory.192.0.2.10`

Use a custom expression on the HttpRequest

Custom expressions enable you to name transactions using getter chain(s) for HttpRequest properties and methods.

1. Set the URI identification option.
2. Click **Apply a custom expression on HttpRequest and use the result in Transaction Names**.
3. Enter your custom expression getter chain as follows:
   - Enclose getter chain(s) inside braces: `${}` .
   - Use getter chain syntax.
   - Use any HttpRequest request attributes or methods.

   For example, consider this URL:

   `http://mystore.example.com/Store/Inventory-Furniture`

   The following custom expression uses two getter chains:
   - The first getter chain fetches the URL, splits it on the dash character ("-"), and uses the second string in the array.
   - The second getter chain fetches the `HttpRequest.UserAgent` property.
   - The literal dash character "-" separates the two getter chains.

   ```
   ${Url.ToString().Split(Char[]/-).[2]}-${UserAgent}
   ```

   The result is the following business transaction name:

   `Furniture-Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko`

Custom Match Rules for ASP.NET Transactions

Custom match rules provide greater flexibility for transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name.

For steps to access the **Custom Match Rules** pane, see To create custom match rules for. NET entry points .

Create an ASP.NET custom match rule

1. In the **Custom Match Rules** pane, click the plus symbol (**+**) to add an entry point.
2. Click **ASP.NET** in the dropdown list. Click **Next**.

3. Name the **New Business Transaction Match Rule**.
   - AppDynamics uses the rule **Name** to name the BT.
   - The Controller enables the rule by default. Disable it later if needed.
   - Set the **Priority** for the match rule. AppDynamics applies higher priority rules first.
4. Set one or more of the following match criteria. When AppDynamics detects a requests matching your specified criteria, it identifies the request using your custom name.

   Method: Match on the HTTP request method, GET, POST, PUT or DELETE.

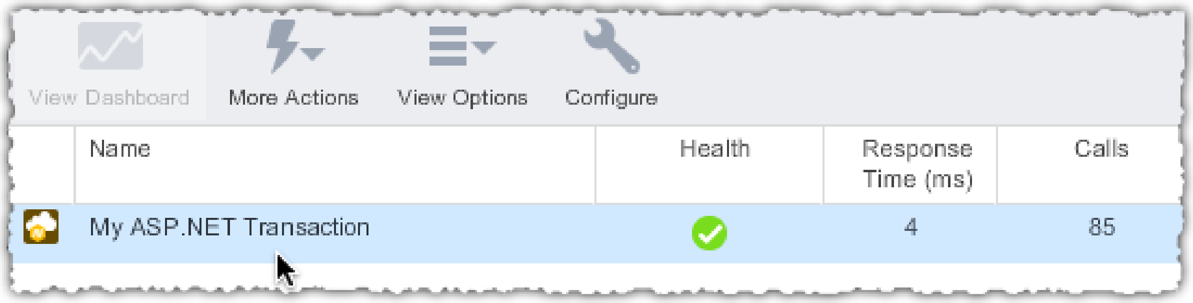
   WIth automatic discovery for ASP.NET transactions enabled, configuring the match on GET or POST causes the the agent to discover both GET and POST requests. If you only want either GET or POST requests for the transaction, consider the following options:

   - Disable automatic discovery for ASP.NET transactions.
   - Create an exclude rule for the method you don’t want: GET or POST.

   URI: Set the conditions to match for the URI.
   - For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
   - Optionally click the gear icon to set a NOT condition.
   - You must set an URI match condition in order to use transaction splitting.

   HTTP Parameter: Match on HTTP parameter existence or a specific HTTP parameter value.
   Header: Match on a specific HTTP header's (parameter's) existence or a specific HTTP header value.
   Hostname: Match on the server host name. Optionally click the gear icon to set a NOT condition.
   Port: Match on the server port number. Optionally click the gear icon to set a NOT condition.
   Class Name: Match on the ASP.NET class name. Optionally click the gear icon to set a NOT condition.
   Cookie: Match on cookie existence or a specific a specific cookie value.

5. Click **Create Custom Match Rule**.
   The rule appears in the **Custom Match Rule** list. The business application or tier you customized displays a green check in the **Select Application or Tier** pane.
   After the agent receives the updated configuration, it discovers the new business transaction and displays it in the Business Transactions List.



	Name	Health	Response Time (ms)	Calls
	My ASP.NET Transaction	✓	4	85

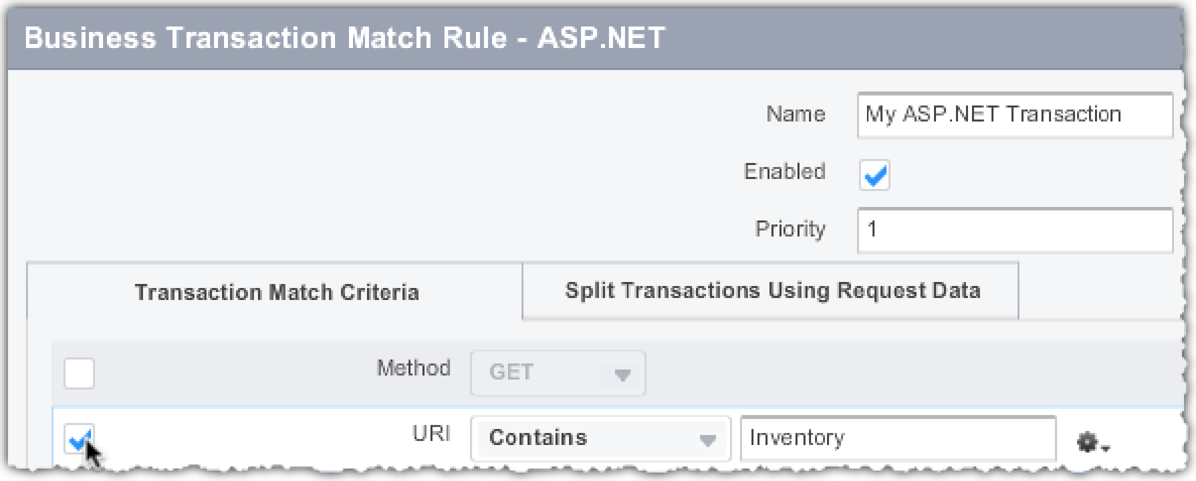
Split custom ASP.NET transactions

AppDynamics lets you further refine ASP.NET custom transaction names using transaction splitting. See Transaction Splitting for Dynamic Discovery.

1. Create a custom match rule. To use transaction splitting, you must specify URI match criteria.
2. Click **Split Transactions Using Request Data**.
3. Click the splitting option to use.
   The transaction splitting options work the same as the methods described in the previous sections:
   Identify transactions using URI segments
   Identify Transactions Using Headers, Cookies, and Other Parts of HTTP Requests

   For example, consider the following URL:
   `http://example.com/Store/Inventory?category=electronics`

   Configure the custom match rule to match on the "URI contains Inventory".

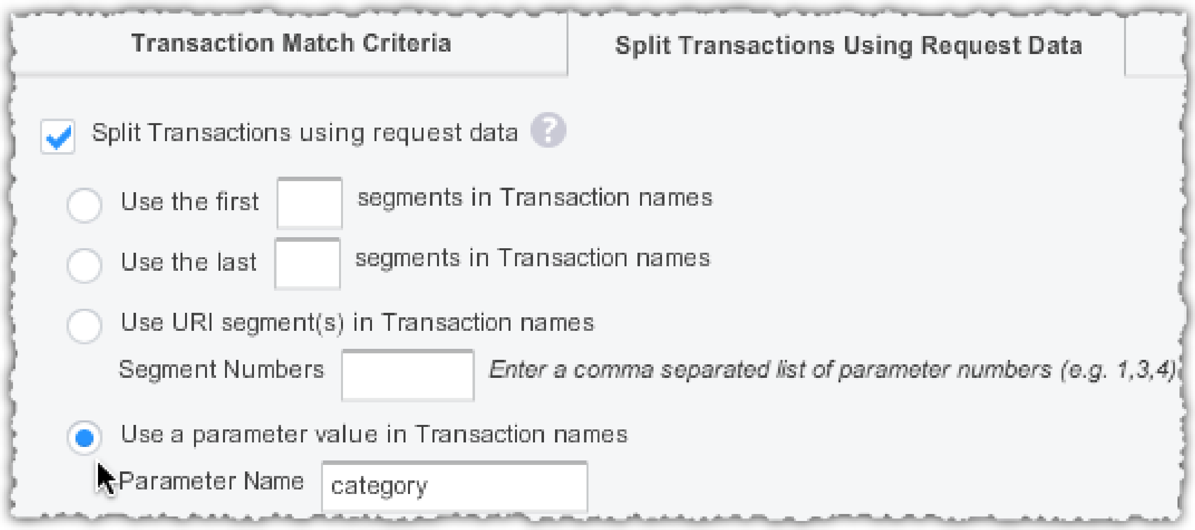

Business Transaction Match Rule - ASP.NET

Name: My ASP.NET Transaction
Enabled: ✓
Priority: 1

Transaction Match Criteria
Split Transactions Using Request Data

Method: GET
✓ URI: Contains Inventory

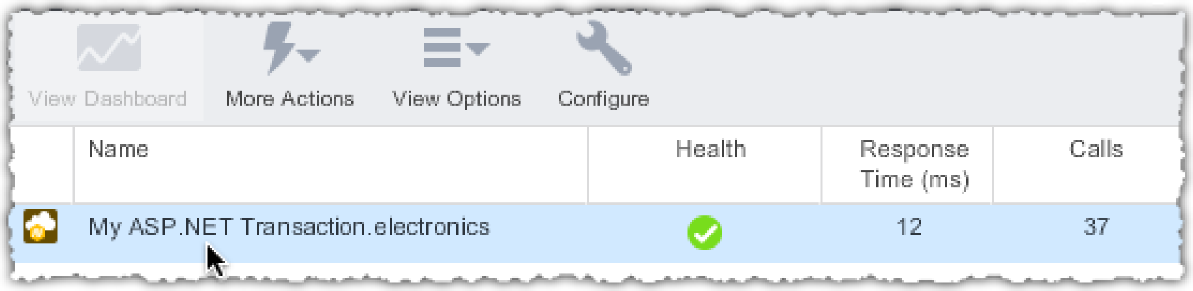
Split the transaction on the category parameter.


Transaction Match Criteria
Split Transactions Using Request Data

✓ Split Transactions using request data

- Use the first ___ segments in Transaction names
- Use the last ___ segments in Transaction names
- Use URI segment(s) in Transaction names
  Segment Numbers ___ *Enter a comma separated list of parameter numbers (e.g. 1,3,4)*
- ● Use a parameter value in Transaction names
  Parameter Name: category

4. Click **Save**.
   After the agent receives the updated configuration, it discovers the new business transaction and displays it in the Business Transactions List.



	Name	Health	Response Time (ms)	Calls
	My ASP.NET Transaction.electronics	✓	12	37

WCF Entry Points

On this page:

- Automatic Naming for WCF Transactions
- Custom Match Rules for WCF Transactions

Related pages:

- Configure Business Transaction Detection for .NET

Microsoft Windows Reference:

- Windows Communication Foundation

AppDynamics automatically detects entry points for client requests to Windows Communication Foundation (WCF) services. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. For information on originating tiers, see Configure Business Transaction Detection.

The .NET Agent detects async entry points for the following patterns:

- Task-based asynchronous operations
- *New in 4.0.2*, IAsyncResult Begin-End Asynchronous pattern

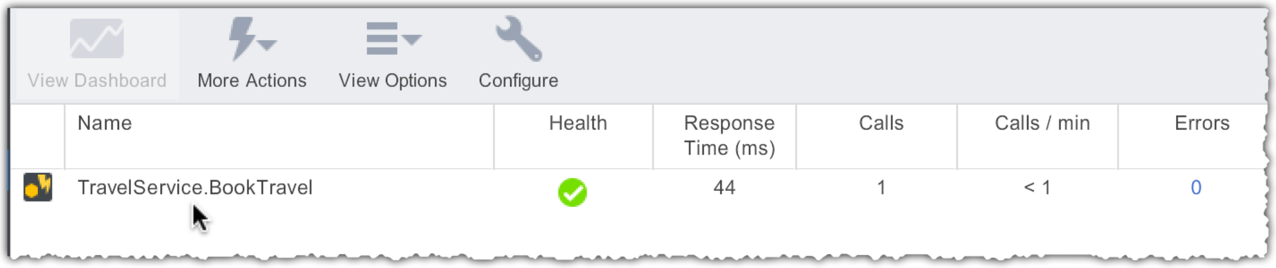
When WCF calls execute on a downstream tier, AppDynamics includes them as part of the business transaction from the originating tier.

Automatic Naming for WCF Transactions

By default, the AppDynamics auto-detection naming scheme identifies all WCF transactions using the service name and operation name:

```
ServiceName.OperationName
```

For example, a web service for a travel website books reservations from client front ends.



	Name	Health	Response Time (ms)	Calls	Calls / min	Errors
	TravelService.BookTravel	✓	44	1	< 1	0

You can rename or exclude automatically discovered transactions from the Business Transactions List.

Custom Match Rules for WCF Transactions

Custom match rules provide flexibility for WCF transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name.

For steps to access the **Custom Match Rules** window, see "To create custom match rules for .Net entry points" on Configure Business Transaction Detection for .NET.

Create a WCF custom match rule

1. In the **Custom Match Rules** window, click the plus symbol (**+**) to add an entry point.
2. Click **WCF** in the dropdown list. Click **Next**.
3. Name the **New Business Transaction Match Rule**.
   - AppDynamics uses the rule **Name** to name the BT.
   - The Controller enables the rule by default. Disable it later if needed.
   - Set the **Priority** for the match rule. AppDynamics applies higher priority rules first.
4. Set one or more of the following match criteria. When AppDynamics detects a requests matching your specified criteria, it identifies the request using your custom name.
   Web Service Name
   - For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
   - Optionally use the gear dropdown to set a NOT condition.

   Operation Name
   - For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
   - Optionally use the gear dropdown to set a NOT condition.

   For example, you could report all operations for a TravelService, such as SearchTravel and BookTravel, as one business transaction.

5. Click **Create Custom Match Rule**.
   The rule appears in the **Custom Match Rule** list. The business application or tier you customized displays a green check in the **Select Application or Tier** pane.

After the agent receives the updated configuration, it discovers the new business transaction and d isplays it in the Business Transactions List.

ASP.NET Web Service Entry Points

On this page:

Related pages:

- Configure Business Transaction Detection for .NET

AppDynamics automatically detects entry points for client requests to ASP.NET web services. If the request occurs on an originating tier, the method or operation marks the beginning of a business transaction and defines the transaction name. For more information on originating tiers, see Configure Business Transaction Detection for .NET.

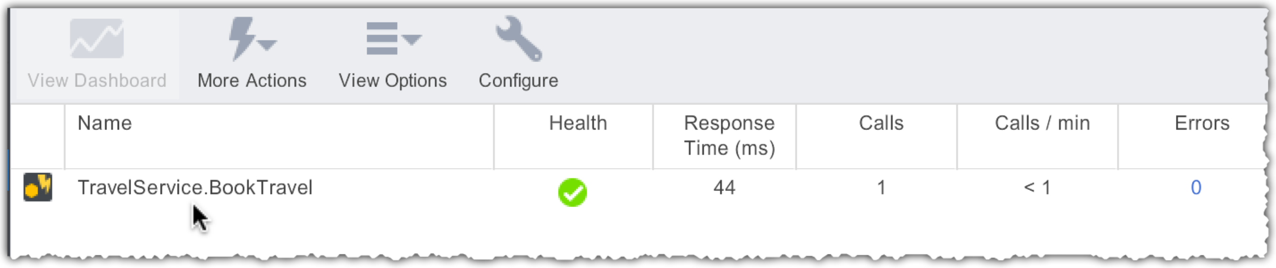
When web service calls execute on a downstream tier, AppDynamics includes them as part of the business transaction from the originating tier.

Automatic Naming for Web Service Transactions

By default, the AppDynamics auto-detection naming scheme identifies all web service transactions using the service name and operation name:

```
ServiceName.OperationName
```

For example, a web service for a travel website books reservations from client front ends.



Name	Health	Response Time (ms)	Calls	Calls / min	Errors
TravelService.BookTravel	✓	44	1	< 1	0

You can rename or exclude automatically discovered transactions from the business transaction list.

Custom Match Rules for ASP.NET Web Service Transactions

Custom match rules provide flexibility for ASP.NET web service transaction naming. When you define a match rule, AppDynamics uses the rule name for the business transaction name.

For steps to access the **Custom Match Rules** pane, see "To create custom match rules for. NET entry points," on Configure Business Transaction Detection for .NET.

To create an ASP.NET web service custom match rule

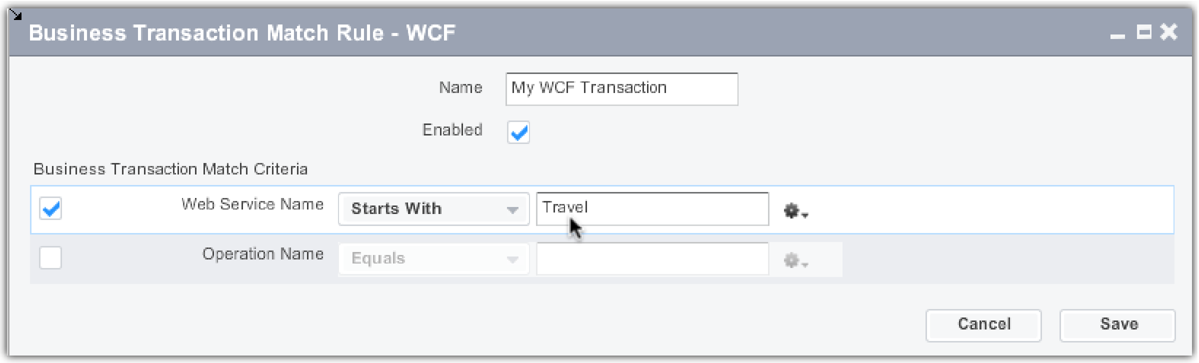
1. In the **Custom Match Rules** pane, click the plus symbol (**+**) to add an entry point.
2. Click **Web Service** in the dropdown list. Click **Next**.
3. Name the **New Business Transaction Match Rule**.
    - AppDynamics uses the rule **Name** to name the BT.
    - The Controller enables the rule by default. Disable it later if needed.
    - Set the **Priority** for the match rule. AppDynamics applies higher priority rules first.

4. Set one or more of the following match criteria. When AppDynamics detects a requests matching your specified criteria, it identifies the request using your custom name.

   Web Service Name
   - For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
   - Optionally use the gear dropdown to set a NOT condition.

   Operation Name
   - For rules on regular expressions for .NET, see .NET Framework Regular Expressions.
   - Optionally use the gear dropdown to set a NOT condition.

   For example, to report all operations for a TravelService, such as SearchTravel and BookTravel, as one business transaction:



Business Transaction Match Rule - WCF

Name: My WCF Transaction

Enabled: ✓

Business Transaction Match Criteria

✓	Web Service Name	Starts With	Travel
	Operation Name	Equals	

5. Click **Create Custom Match Rule**.
   The rule appears in the **Custom Match Rule** list. The business application or tier you customized displays a green check in the **Select Application or Tier** pane.

After the agent receives the updated configuration, it discovers the new business transaction and d isplays it in the Business Transactions list.

Identify MVC Transactions by Controller and Action

Related Pages:

- All Other Traffic Business Transaction
  App Agent Node Properties
  Configure Business Transaction Detection

Microsoft External Reference:

- ASP.NET Routing

*Y*ou can configure the agent to identify transactions by MVC Controller/Action instead of the default naming by URI. For general information on organizing and naming business transactions, see Organize Traffic as Business Transactions.

By default the .NET Agent (agent) identifies ASP.NET MVC business transactions by the request URL or server URI.

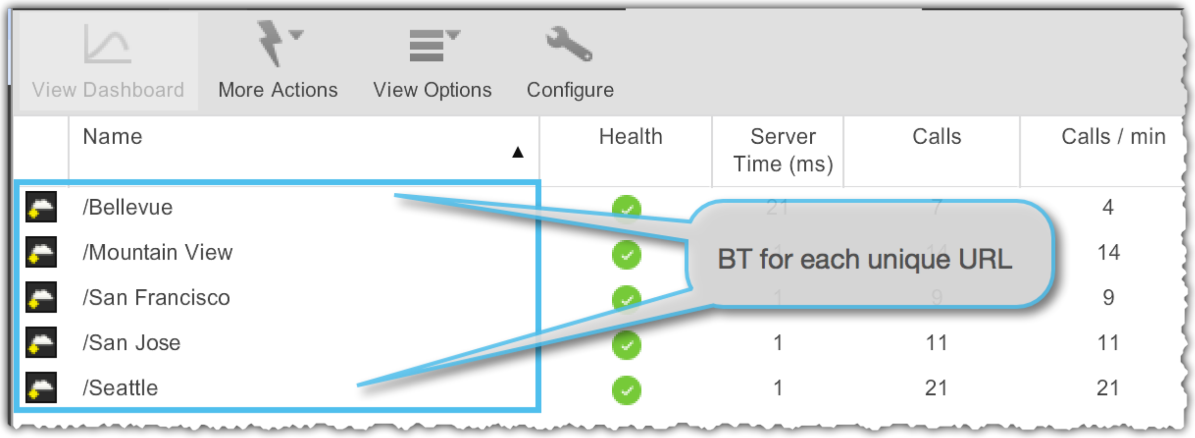
In cases where an application accepts numerous client URLs for a single MVC controller and action, naming business transactions with the client URL can cause several issues including the following:

- The number of business transactions exceeds the limits. See *Business Transaction Limits* in All Other Traffic Business Transaction.
- Most requests wind up in "All other traffic". See All Other Traffic Business Transaction.
- Requests per minute per business transaction is inconsistent.

For example, consider a MVC application that lists store locations. City-specific URLs list the locations for a city:

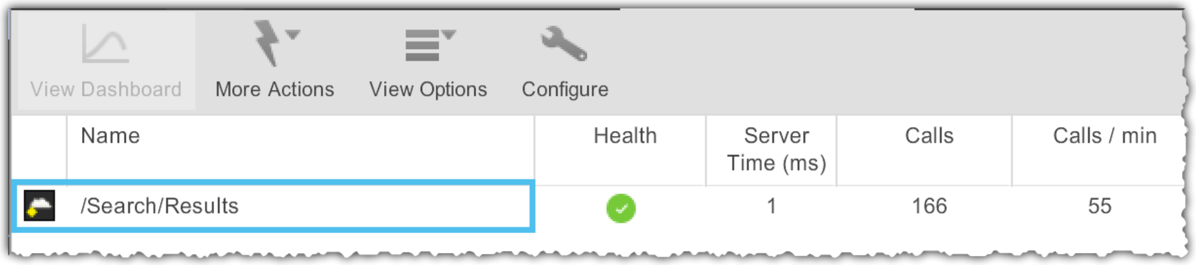
http://myapp.mycompany.com/Bellevue

The business transaction name for this URL defaults to **/Bellevue**. Each request for a unique city generates a business transactions. None of the URIs contain common elements that you can use to configure business transaction names.



	Name ▲	Health	Server Time (ms)	Calls	Calls / min
	/Bellevue	✓	21	7	4
	/Mountain View	✓	1	14	14
	/San Francisco	✓	1	9	9
	/San Jose	✓	1	11	11
	/Seattle	✓	1	21	21

In the web application, all city location searches, such as /Bellevue, map to the *Results* action of the *Search* controller. After you configure the agent to name transactions by controller and action, the agent identifies the business transaction as **Search/Results**.



	Name	Health	Server Time (ms)	Calls	Calls / min
	/Search/Results	✓	1	166	55

The Search/Results business transaction combines search requests for all cities into one transaction.

To configure the agent to identify MVC transactions as Controller/Action

- Register the **aspdotnet-mvc-naming-controlleraction** node property. The node property works for MVC 3, MVC 4 and WebAPI transactions.

For instructions on how to register a new node property, see App Agent Node Properties.

Name: aspdotnet-mvc-naming-controlleraction
Description: Identfy ASP.NET MVC Business Transactions as Controller/Action.
Type: Boolean

Value: True

The default value is **False**.

After the agent registers traffic with a business transaction named for the Controller/Action and after traffic to the old business transactions named for the client URL diminishes, delete the old business transactions.

Business Transaction Naming Convention

If you use Areas to organize your MVC application, the agent includes the Area name in the business transaction name:

```
/Area name/Controller name/Action name
```

For example, if your travel application has separate Areas for hotel, airfare, and car rentals:

/Hotel/Search/Results

Otherwise the agent names the transaction as follows:

```
/Controller name/Action name
```

Getter Chains in .NET Configurations

On this page:

Related pages:

- Configure Business Transaction Detection
- Configure Data Collectors

Use getter chains to add basic programmatic functions to AppDynamics data collection and business transaction configuration. This topic describes the getter chain syntax for the .NET Agent and provides examples to help you get started.

Configurations that support getter chains include:

- Data collectors
- ASP.NET transaction naming
- HTTP backend naming

If a getter chain calls a heavy processing method, such as one that makes numerous SQL calls, the getter chain may degrade performance for both the application and the .NET Agent. When possible, use getter chains only with simple `Get()` methods.

For example, a simple getter that returns a property for a class, such as `MyUser.GetName()`:

```
public class MyUser
{
private String Name {get; set;}
private String Url;
public GetUrl(){
 return this.Url;
 }
}
```

Separator Characters in Getter Chains

The following special characters are used as separators:

- comma (`,`) for separating parameters
- forward slash (`/`) for separating a type declaration from a value in a parameter
- dot (`.`) for separating the methods and properties in the getter chain

If a slash or a comma character is used in a string parameter, use the backslash (`\`) escape character. Except in the case of `type/value`.

If a literal dot is used in a string parameter, use the backslash escape character before the dot: `\.`

Getter Chain Use Cases

The following examples illustrate the AppDynamics getter chain syntax for common scenarios in .NET applications.

Declare parameter types

- The following example demonstrates how to declare the parameter types to resolve the overloaded `Susbstring()` method. The forward slash serves as the type separator.

  ```
  GetAddress(appdynamics, sf).Substring(int/0, int/10)
  ```

  For instance, if `GetAddress(appdynamics, sf)` returns "303 2nd St, San Francisco, CA 94107," then the agent returns "303 2nd St" for the full getter chain.
- The following example shows how to declare the parameter types for a user-defined method that takes a float parameter, a boolean parameter, and an integer parameter. The forward slash serves as the type separator.

  ```
  GetAddress(appdynamics, sf).MyMethod(float/0.2, boolean/true,
  boolean/false, int/5)
  ```

When you declare a type using the forward slash in a type declaration (`type/char`), don't escape the value after forward slash.

Escape a character

- The following example shows the how to escape the dot in the string parameter.

  ```
  GetAddress().GetParam(a\.b\.c\.)
  ```

  The agent executes `GetParam("a.b.c.")` on the result of `GetAddress()` and returns the value of of the parameter.

- In the following example, the first dot is part of the string method parameter, which requires an escape character. The second and third dots don't require an escape character because they are part of the getter chain syntax.

  ```
  GetUser(suze\.smith).GetGroup().GetId()
  ```

Access indexed properties and dictionary values

- The following example returns the value for the key "suze.smith" using the implied getter.

  ```
  UserDictionary.get_Item(suze\.smith)
  ```

Split a character array

- The following example splits a URL on the forward slash character, which is escaped. It returns the fourth item in the array.

  ```
  GetUrl().Split(char[]//).[3]
  ```

  The agent returns "Search" when it applies the getter chain to the following URL:

  http://howdyworld.example.com/Search/Airfare
- The following example splits on the forward slash character, then returns the length property of the resulting array.

  ```
  GetUrl().Split(char[]//).Length
  ```

- The following example illustrates a transaction splitting rule for URIs that use a semicolon delimiter. The getter chain splits on the forward slash, then splits the fourth element on the semicolon.

  ```
  GetUri().Split(char[]//).[3].Split(char[]/;).[0]
  ```

  The agent returns "create-user" when it applies the getter chain to the following URL:

  http://HowdyWorld.example.com/create-user;sessionid=BE7F31CC0235C796BF8C6DF376

6A1D00?act=Add&uid=c42ab7ad-48a7-4353-bb11-0dfeabb798b5

Getter Chains for HttpRequest Objects

The .NET Agent requires special syntax for getter chains on System.Web.HttpRequest objects in ASP.NET WebForms, MVC, and MVC WebAPI applications. Use the following syntax to delineate the boundaries of the getter chain:

```
${myobject.myproperty}
```

For example, to determine the user principal:

```
${Context.User.Identity.Name}
```

Places to use this syntax include:

- HTTP Request Data Collectors.

Configure Service Endpoints for .NET

On this page:

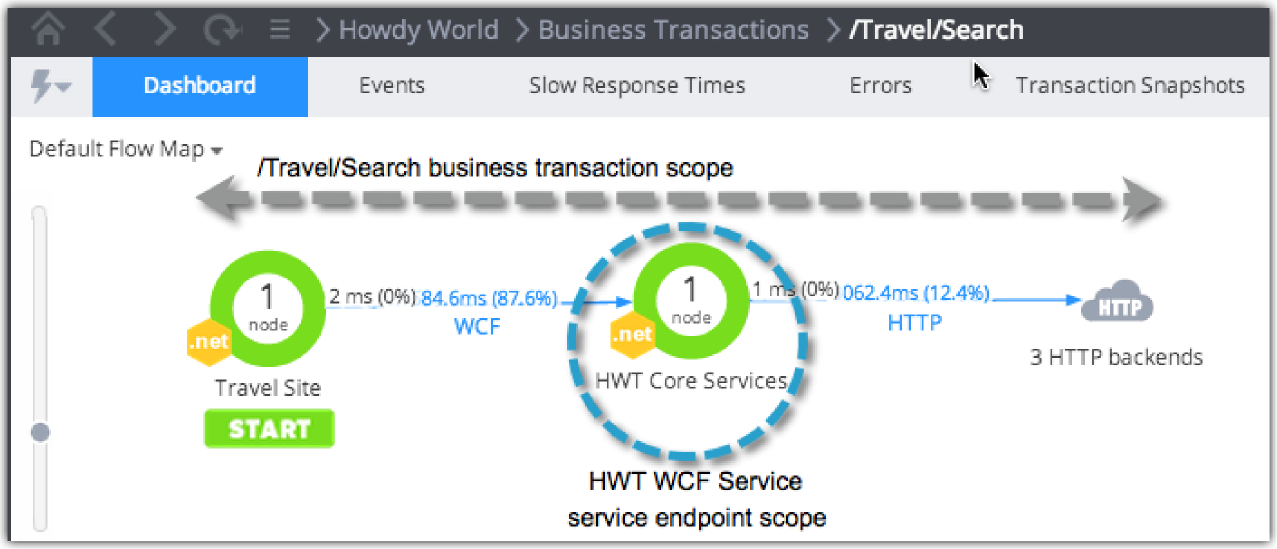
- Tips for Defining .NET Service Endpoints

Related pages:

- Monitor Service Endpoints

Configure service endpoints to monitor the performance of an entry point for a service or application on a single tier.

For example, consider a Travel site that has an ASP.NET MVC front end. When a customer searches, the /Travel/Search business transaction originates on the MVC tier that calls a WCF service. The WCF service in turn calls external travel sites, aggregates the results, and returns the search results to the MVC front end.



If you are solely responsible for the HWT Core Services tier or the WCF service, a service endpoint lets you focus on the performance metrics that apply to your tier.

Because you decide which endpoints are important, the .NET Agent doesn't automatically discover service endpoints. Define the service endpoint as you would a custom match rule for a business transaction. See Monitor Service Endpoints and Configure Business Transaction Detection.

You can create service endpoints for the following entry point types:

ASP.NET
Message Queues
POCO
WCF
Web Service

The .NET Agent doesn't support service endpoints for MSMQ.

After you configure a service endpoint, you can monitor it on the service endpoint dashboard that shows KPIs and transaction snapshots where the service endpoint executed. You can also use the metric browser to analyze service endpoint performance. See Monitor Service Endpoints.

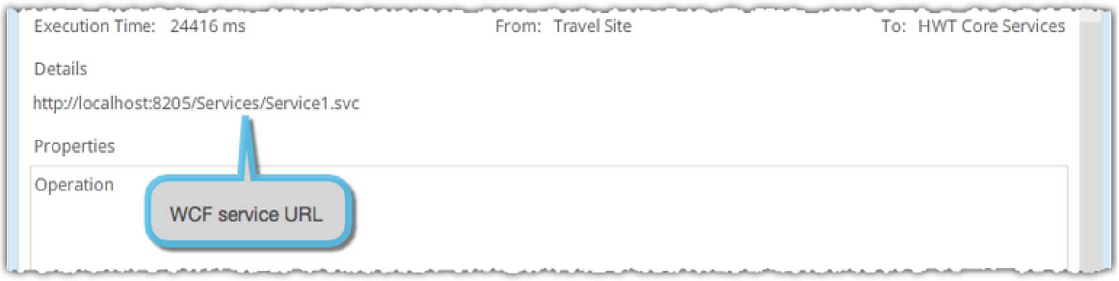
Tips for Defining .NET Service Endpoints

Although the .NET Agent doesn't automatically discover service endpoints, frequently you want to monitor service endpoints that match automatically discovered entry points in a business transaction:

- You can create service endpoints on the originating entry point for a business transaction.
- For ASP.NET service endpoints, you can use the URL.

For example, in the Travel Site tier above: URL matches "`/Travel/Search`"

- For WCF and ASP.NET web service entry points, you can use an existing transaction snapshot to find the URL for the service.
  1. Open a full transaction snapshot for the business transaction.
  2. From the upstream tier, click the exit call link, either **WCF** or **Web Service**. The exit call window shows the URL for the web service.



Execution Time: 24416 ms From: Travel Site To: HWT Core Services

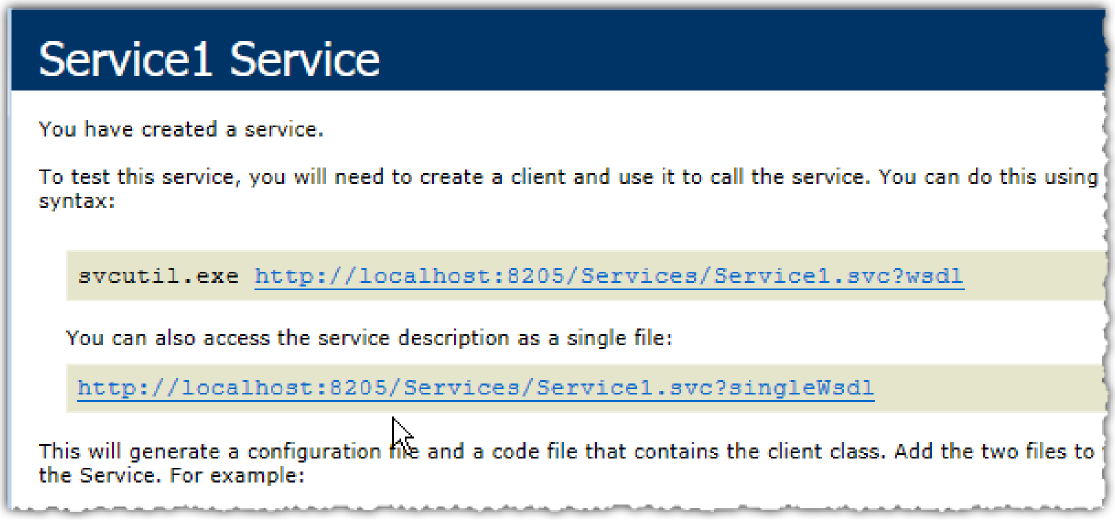
Details

http://localhost:8205/Services/Service1.svc

Properties

Operation

3. Open the service URL in a browser.
4. On the service page, the url to access the wsdl file.



Service1 Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using syntax:

```
svcutil.exe http://localhost:8205/Services/Service1.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:8205/Services/Service1.svc?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to the Service. For example:

5. The `service name element shows the service name.`

```
<wsdl:service name="Service1">
```

- Define POCO service endpoints exactly as you would a POCO custom match rule.

Configure Backend Detection for .NET

On this page:

- Custom Exit Points

Related Pages:

- .NET Supported Environments
- Configure Backend Detection
- Configure Custom Exit Points
- Monitor Async Backends for .NET

To review general information about monitoring databases and remote services (collectively known as backends) and for an overview of backend configuration see Backend Monitoring.

View and Revise Backend Discovery Rules

To view the discovery rules the .NET Agent uses to automatically discover backends:

1. Click the application or tier.
2. In the left navigation pane, click **Configure -> Instrumentation**.
3. Click the Backend Detection tab.
4. Click the .NET - Backend Detection tab.
   The .NET - Backend Detection tab lists the default configurations for Automatic Backend Discovery
5. In the Automatic Backend Discovery list, click the backend type to view.
   A configuration summary appears on the right.

   For example, AppDynamics enables automatic discovery for HTTP backends by default and names them for the host and port.

Each automatically discovered backend type has a default discovery rule and a set of configurable properties. For instructions to revise backend discovery rules, see Configure Backend Detection.

ADO.NET Backends

The .NET Agent automatically discovers ADO.NET data providers implementing standard Microsoft interfaces as database backends. For a complete list, see "Supported ADO.NET Clients" on .NET Supported Environments.

Because the ADO.NET API is interface-based, by default AppDynamics instruments all ADO.NET database providers that implement these interfaces.

For database identification, AppDynamics uses information from the ADO.NET connection string. The connection string specifies the server address and schema or the local file name. Most connection strings are formatted according to well-known rules that can be parsed and distilled to a database name. However, because there is no standard on the connection string, it is up to the ADO.NET provider implementer to choose the format.

For some providers, AppDynamics may fail to parse the connection string. In these cases, the .NET agent uses the complete connection string minus any user password. The property is labeled **ADO.NET connection string** and the value shows the connection string minus any user password.

For example, the agent names following database backend using the connection string pattern <datasource name>-<database name>:

.\SQLEXPRESS-HowdyWorldDB

ADO.NET configurable properties

You can enable or disable the use of the following properties for ADO.NET exit points:

Configurable Properties	**Property Used by Default in Detection and Naming**	**Description**
Host	Yes	data source or database server
Database	Yes	database name
Vendor	No	type of the client-side ADO.NET library
Connection String	No	full connection string with password filtered out
Port	No	port number

Directory Service Backends

The .NET Agent automatically discovers exit calls to directory services that use the System.DirectoryServices.Protocols (S.DS.P) libraries.

The agent names the backend for the server name. If the agent is unable to derive the server name from the request, it constructs a name using Domain Component (DC) values.

For example:

activedirectory.example.com

HTTP Backends

AppDynamics automatically detects HTTP exit points (backends) that use the Microsoft HTTP client. The default HTTP automatic discovery rule uses the URL property. From the enabled properties AppDynamics derives a display name using the URL.

For example:

http://api.example.com:8989/searchfares

HTTP configurable properties

You can enable or disable the use of the following properties for HTTP exit points:

Configurable Properties	**Property Used by Default in Detection and Naming**	**Description**
Host	No	HTTP host
Port	No	HTTP port number

URL	Yes	full URL
Query String	No	HTTP parameters/query string

Message Queue Backends

By default, AppDynamics automatically detects and identifies many message queue exit points. For a list of the supported message-oriented middleware products, see Supported Remote Service Detection for .NET.

The default queue automatic discovery rule uses the destination property.

For example:

HowdyWorldQueue$\HWT_MQ_Server1_MsgQ

Message Queue Configurable Properties

In general, the properties listed below are used for queue exit points. However, each message-oriented product is different and there may be variations in the properties or their names.

Configurable Properties	**Property Used by Default in Detection and Naming**	**Description**
Host	No	queue server name
Destination	Yes	name of topic or queue
Destination Type	No	queue or topic
Vendor	No	vendor from the client library

Note
See Monitor RabbitMQ Backends for .NET for backend naming configuration options.

See Monitor MSMQ Backends for .NET for MSMQ and NServiceBus configuration.

.NET Remoting

AppDynamics automatically detects and identifies remoting exit points (backends) when an application uses .NET remoting.

The default remoting automatic discovery rule uses the URL property.

For example:

tcp://remoting.example.com:8648/MovieTicketBooking

See Enable Correlation for .NET Remoting to configure downstream correlation.

.NET Remoting configurable properties

You can configure the use of the following property for .NET Remoting exit points:

Configurable Properties	Property Used by Default in Detection and Naming	Description
URL	Yes	full URL

WCF Backends

AppDynamics automatically detects and identifies WCF exit points (backends) when an application uses the WCF client library. The default WCF automatic discovery rule uses the remote address property. The agent uses the enabled properties to derive a display name using the remote address.

For example:

> http://wcf.example.com:8205/Services/Service1.svc

WCF configurable properties

You can enable or disable the use of the following properties for WCF exit points:

Configurable Properties	Property Used by Default in Detection and Naming	Description
Remote Address	Yes	URL minus the query, fragment and user information (name and password)
Operation Contract	No	WCF operation name
URL	No	full URL
Host	No	host portion of URL
Port	No	port number if present in the URL, otherwise protocol default
SOAP Action	No	for web service calls, the SOAP action

.NET Web Services Backends

By default, AppDynamics automatically detects and identifies web services exit points (backends) when an application uses the Microsoft Web Services client library. The default web services automatic discovery rule uses the URL property. From the enabled properties AppDynamics derives a display name using the URL.

For example:

http://webservice.example.com:8105/Services/Service1.asmx

Web Services configurable properties

You can enable or disable the use of the following properties for Web Services exit points:

Configurable Properties	**Property Used by Default in Detection and Naming**	**Description**
Service	No	web service name
URL	Yes	full URL
Operation	No	web service operation name
Soap Action	No	SOAP action

All Other Traffic

For information on All Other Traffic transactions, see All Other Traffic Business Transaction.

Custom Exit Points

If you want to monitor a backend that isn't listed under "Remote Service Detection" on .NET Supported Environments, then configure a custom exit point.

Monitor Async Backends for .NET

On this page:

Related pages:

- Configure Backend Detection for .NET
- Using Asynchronous Methods in ASP.NET 4.5

This topic describes the .NET asynchronous programming patterns the agent detects and provides instruction on how to use AppDynamics Controller features that represent asynchronous transactions.

For more information about remote services and exit points, see Monitor Remote Services and Co nfigure Backend Detection for .NET.

Asynchronous Backend Calls in .NET

Developers use asynchronous programming patterns to create scalable, more performant applications. Microsoft .NET lets you designate methods as asynchronous tasks. The .NET runtime releases resources for asynchronous methods while tasks complete. When task processing finishes, the runtime calls back to the originating asynchronous method so the method may continue processing.

Because tasks may execute in parallel, AppDynamics sometimes represents asynchronous activity differently from synchronous activity in the Controller.

Supported asynchronous exit point patterns

The agent automatically discovers the following asynchronous programming patterns for HTTP, Web Service, and WCF exit points:

- Microsoft .NET 4.5 **async** and **await** keywords. See Asynchronous Programming with Async and Await.

New in 4.0.5, the agent automatically discovers ADO.NET async and await exit calls.

AppDynamics separates .NET async backend tracking from thread correlation. To configure thread correlation, see Enable Thread Correlation for .NET.

Identify Asynchronous Backend Calls in Dashboards

When AppDynamics detects asynchronous exit points in an application it displays a dotted line labelled "async" for the flow in the dashboards. Because they may execute simultaneously the Controller doesn't display percentage value of the end-to-end transaction time for asynchronous calls.

Troubleshoot Asynchronous Calls in Transaction Snapshots

Transaction snapshots include several features to help you discover problem areas in business transactions that use asynchronous methods. For an overview of transaction snapshots, see Trans action Snapshots.

Transaction Snapshot Flow Map

The Transaction Snapshot Flow Map graphically represents the business transaction. It displays the user experience, execution time, and timestamp of the transaction. The flow map also provides details of the overall time that is spent in a particular tier and in database and remote service calls. The **async** label indicates asynchronous calls.

Snapshot Execution Waterfall View

The transaction Snapshot Execution Waterfall View shows a timeline representation of the end-to-end execution of the business transaction. Synchronous and asynchronous processes appear on a bar diagram illustrating their relative execution time arranged in chronological order.

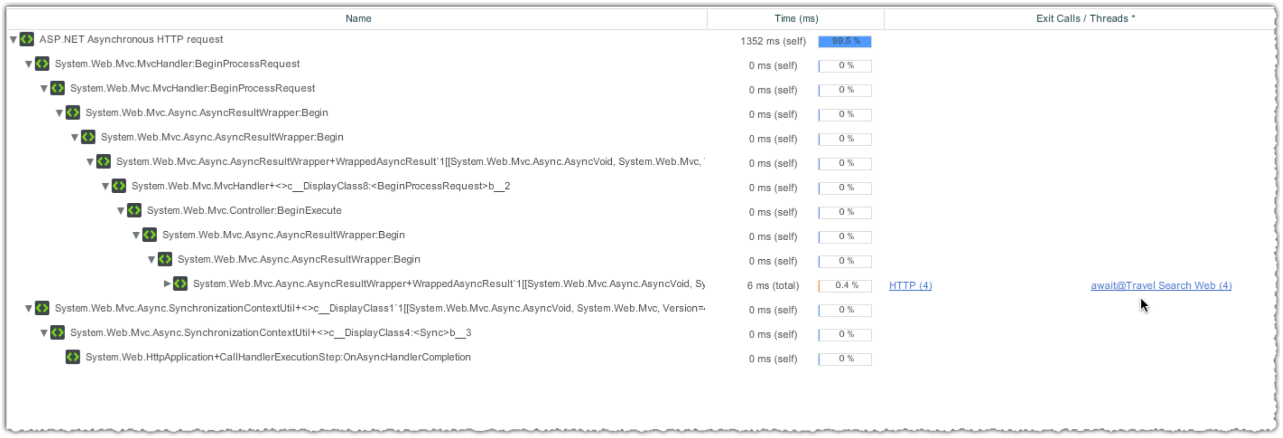
The waterfall view enables you to visually identify which processes are running the longest. Double-click a bar to drill down to a call graph and investigate problematic code.

Call Graph

When the agent detects asynchronous exit points, it displays an **await** link in the **Exit Calls/Threads** column of the call graph in the Call Drill Down. Click the link to display a list of the asynchronous calls. The format of the await link is as follows:

```
await@<tier name>
```

For example, the Travel Search Web tier makes asynchronous calls to the internal customer management system and to provider backends. The link shows as "await@Travel Search Web".

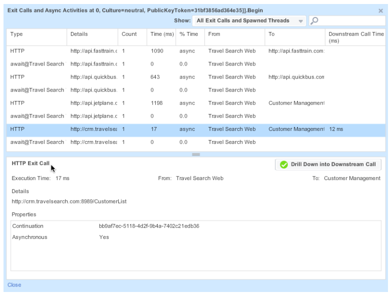


Name	Time (ms)		Exit Calls / Threads *	
▼ ASP.NET Asynchronous HTTP request	1352 ms (self)	99.5 %		
▼ System.Web.Mvc.MvcHandler:BeginProcessRequest	0 ms (self)	0 %		
▼ System.Web.Mvc.MvcHandler:BeginProcessRequest	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.AsyncResultWrapper:Begin	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.AsyncResultWrapper:Begin	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.AsyncResultWrapper+WrappedAsyncResult`1[[System.Web.Mvc.Async.AsyncVoid, System.Web.Mvc,	0 ms (self)	0 %		
▼ System.Web.Mvc.MvcHandler+<>c__DisplayClass8:<BeginProcessRequest>b__2	0 ms (self)	0 %		
▼ System.Web.Mvc.Controller:BeginExecute	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.AsyncResultWrapper:Begin	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.AsyncResultWrapper:Begin	0 ms (self)	0 %		
▶ System.Web.Mvc.Async.AsyncResultWrapper+WrappedAsyncResult`1[[System.Web.Mvc.Async.AsyncVoid, Sy	6 ms (total)	0.4 %	HTTP (4)	await@Travel Search Web (4)
▼ System.Web.Mvc.Async.SynchronizationContextUtil+<>c__DisplayClass1`1[[System.Web.Mvc.Async.AsyncVoid, System.Web.Mvc, Version=	0 ms (self)	0 %		
▼ System.Web.Mvc.Async.SynchronizationContextUtil+<>c__DisplayClass4:<Sync>b__3	0 ms (self)	0 %		
System.Web.HttpApplication+CallHandlerExecutionStep:OnAsyncHandlerCompletion	0 ms (self)	0 %		

The Exit Calls and Async Activities window shows a list of the exit calls and the corresponding **aw ait** continuation calls.

- Exit calls display by type: HTTP, web service, or WCF.
- Continuations/call backs display as `await@tier` name.

For example:
HTTP call to the Customer
Management tier



Exit Calls and Async Activities at 0, Culture=neutral, PublicKeyToken=31bf3856ad364e35]].Begin

Show: All Exit Calls and Spawned Threads

Type	Details	Count	Time (ms)	% Time	From	To	Downstream Call Time (ms)
HTTP	http://api.fasttrain.c	1	1090	async	Travel Search Web	http://api.fasttrain.com:	
await@Travel Search	http://api.fasttrain.c	1	0	0.0	Travel Search Web		
HTTP	http://api.quickbus.	1	643	async	Travel Search Web	http://api.quickbus.con	
await@Travel Search	http://api.quickbus.	1	0	0.0	Travel Search Web		
HTTP	http://api.jetplane.c	1	1198	async	Travel Search Web	Customer Management	
await@Travel Search	http://api.jetplane.c	1	0	0.0	Travel Search Web		
HTTP	http://crm.travelsea	1	17	async	Travel Search Web	Customer Management	12 ms
await@Travel Search	http://crm.travelsea	1	0	0.0	Travel Search Web		

HTTP Exit Call

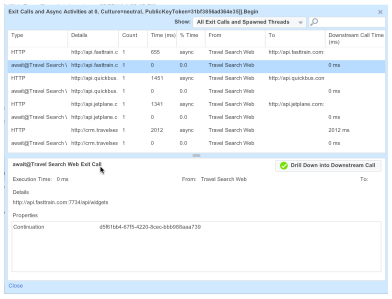
Execution Time: 17 ms From: Travel Search Web To: Customer Management

Details
http://crm.travelsearch.com:8989/CustomerList

Properties

Continuation	bb9af7ec-5118-4d2f-9b4a-7402c21edb36
Asynchronous	Yes

Continuation call back to the
Travel Search tier



Exit Calls and Async Activities at 0, Culture=neutral, PublicKeyToken=31bf3856ad364e35]].Begin

Show: All Exit Calls and Spawned Threads

Type	Details	Count	Time (ms)	% Time	From	To	Downstream Call Time (ms)
HTTP	http://api.fasttrain.c	1	655	async	Travel Search Web	http://api.fasttrain.com:	
await@Travel Search \	http://api.fasttrain.c	1	0	0.0	Travel Search Web		0 ms
HTTP	http://api.quickbus.	1	1451	async	Travel Search Web	http://api.quickbus.con	
await@Travel Search \	http://api.quickbus.	1	0	0.0	Travel Search Web		0 ms
HTTP	http://api.jetplane.c	1	1341	async	Travel Search Web	http://api.jetplane.com:	
await@Travel Search \	http://api.jetplane.c	1	0	0.0	Travel Search Web		0 ms
HTTP	http://crm.travelsea	1	2012	async	Travel Search Web		2012 ms
await@Travel Search \	http://crm.travelsea	1	0	0.0	Travel Search Web		0 ms

await@Travel Search Web Exit Call

Drill Down into Downstream Call

Execution Time: 0 ms   From: Travel Search Web   To:

Details

http://api.fasttrain.com:7734/api/widgets

Properties

Continuation	d5f61bb4-67f5-4220-8cec-bbb988aaa739

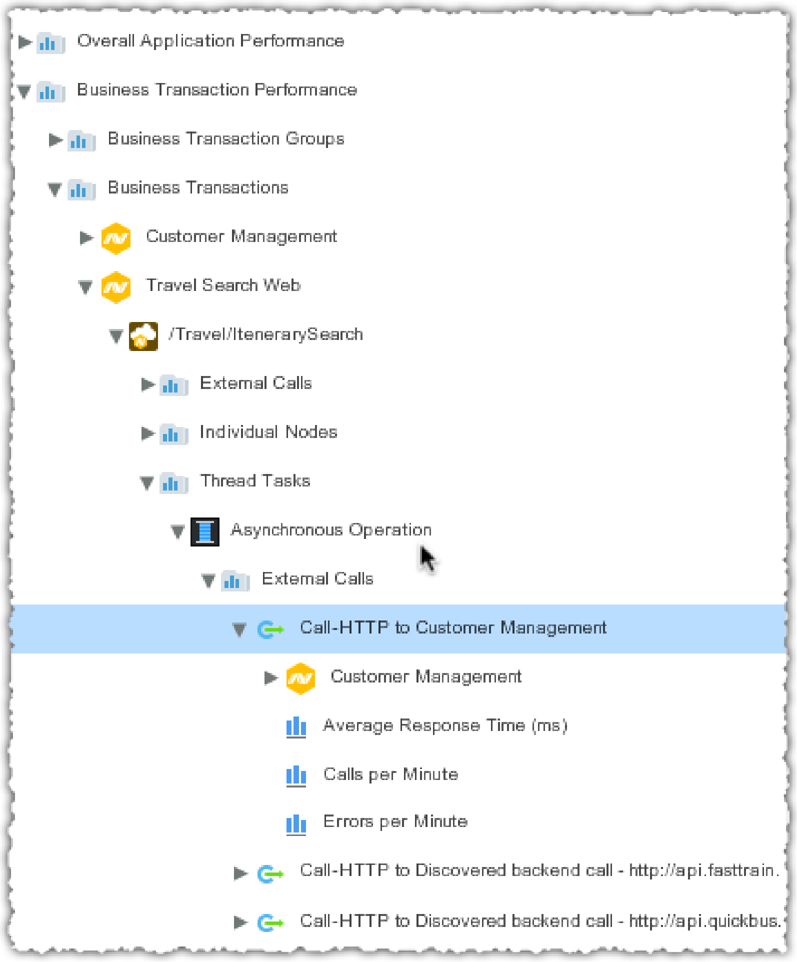
Close

Analyze Asynchronous Activity in the Metric Browser

The Metric Browser displays asynchronous activity in the following places:

- **Business Transaction Performance -> Business Transactions -> tier > business transaction -> Thread Tasks -> Asynchronous Operation -> External Calls**
- **Overall Application Performance -> tier -> Thread Tasks -> Asynchronous Operation -> External Calls**
- **Overall Application Performance -> tier -> Individual Nodes -> node name -> Thread Tasks -> Asynchronous Operation -> External Calls**

For example:


Overall Application Performance
Business Transaction Performance
Business Transaction Groups
Business Transactions
Customer Management
Travel Search Web
/Travel/IteneranySearch
External Calls
Individual Nodes
Thread Tasks
Asynchronous Operation
External Calls
Call-HTTP to Customer Management
Customer Management
Average Response Time (ms)
Calls per Minute
Errors per Minute
Call-HTTP to Discovered backend call - http://api.fasttrain.
Call-HTTP to Discovered backend call - http://api.quickbus.

For more information on how to use the Metric Browser, see Metric Browser.

Monitor MSMQ Backends for .NET

On this page:

Related pages:

- Configure Backend Detection for .NET
- App Agent Node Properties

Remote Service Detection

The AppDynamics .NET Agent (agent) automatically detects MSMQ backends based upon exit calls from instrumented tiers. MSMQ exit points are methods that publish or push messages to a queue.

Before the agent can detect entry points or perform downstream correlation for MSMQ you must define the correlation field and, for multi-threaded architectures like NServiceBus, specify threading architecture.

To view MSMQ or NServiceBus in the list of backends, click **Servers->Remote Services** in the left navigation pane.

Exit Points and Backend Naming

The agent names the queue for the queue name returned from the system.

Entry Points

To enable downstream correlation for MSMQ, you must configure the agent as follows:

Define the MSMQ correlation field

Register the **msmq-correlation-field** app agent node property on both the publishing tier and on the receiving tier. Specify the field where the agent will write and read correlation information.

The agent supports using "Label" or "Extension." In general, "Extension" is the best choice because it is a larger field. However, the NServiceBus implementation of MSMQ uses the "Extension" field, so for NServiceBus you must use "Label."

Choose a field not in use by your queue implementation. If your implementation uses both the "Label" and "Extension" fields, downstream correlation is not currently possible.

See "msmq-correlation-field" on App Agent Node Properties Reference. For instructions to register a node property, see App Agent Node Properties.

Specify the threading architecture

If your queue implementation uses a multi-threaded architecture, you must register the **msmq-single-threaded** app agent node property to specify the threading architecture. Set msmq-single-threaded to "False" for NServiceBus or other multi-threaded implementations of MSMQ. See "msmq-single-threaded" on App Agent Node Properties Reference. For instructions to register a node property, see App Agent Node Properties.

The threading architecture dictates how the agent calculates the call timing for the message queue:

- For single-threaded queues, the agent calculates the call time between receive requests. The call time begins at the end of the first receive method and ends when the next receive method starts.
  In the following example, the length of the `ProcessMessage()` method:

  ```
  MessageQueue messageQueue;
  for(;;)
  {
          var message = messageQueue.Receive();
          ProcessMessage(message);
  }
  ```

- For multi-threaded queues, the agent does not capture timing.

Monitor Oracle Backends for .NET with AppDynamics for Databases

On this page:

Related pages:

- Introduction to AppDynamics for Databases
- Monitor Databases

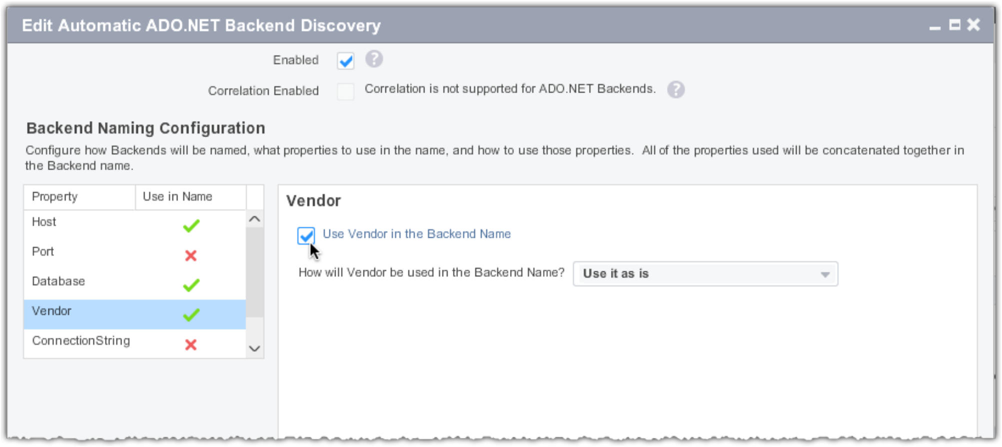
Oracle ODP.NET database backends integrate with AppDynamics for Databases. This topic covers how to configure integration and an introduction to the enhanced features.

Prerequisites

Before you can take advantage of the integration for Oracle backends with AppDynamics for Databases, you must perform the following setup:

1. Install AppDynamics for Databases and add a collector for your Oracle database. See Install AppDynamics for Databases and Add an AppDynamics for Databases Collector.
2. Enable integration with AppDynamics for Databases in the Controller. See Integrate with AppDynamics for Databases.

3. Log off from the Controller, then log on again.
4. Enable the Vendor property for ADO.NET backend naming. For more information see Configure Backend Detection for .NET.



Edit Automatic ADO.NET Backend Discovery

Enabled ✓

Correlation Enabled — Correlation is not supported for ADO.NET Backends.

Backend Naming Configuration

Configure how Backends will be named, what properties to use in the name, and how to use those properties. All of the properties used will be concatenated together in the Backend name.

Property	Use in Name
Host	✓
Port	✗
Database	✓
Vendor	✓
ConnectionString	✗

Vendor

✓ Use Vendor in the Backend Name

How will Vendor be used in the Backend Name? Use it as is

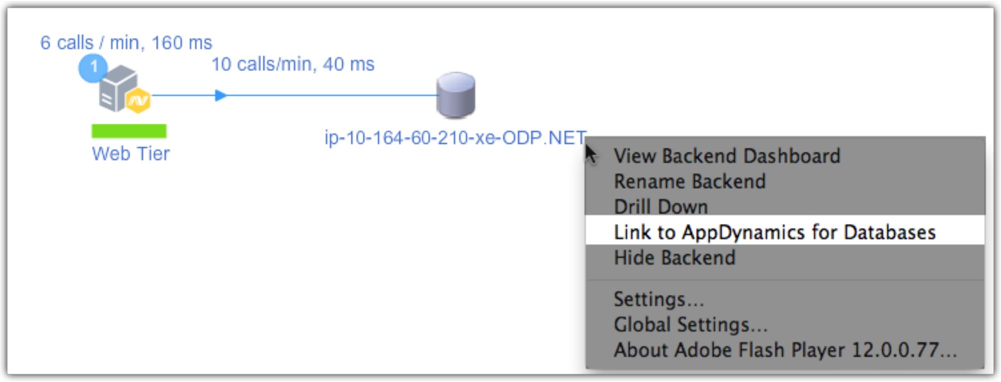
> ⓘ AppDynamics for Databases requires the Vendor property to identify the Oracle database.

Monitor Oracle Database Backends

After you configure integration, AppDynamics enables links from the Controller to AppDynamics for Databases.

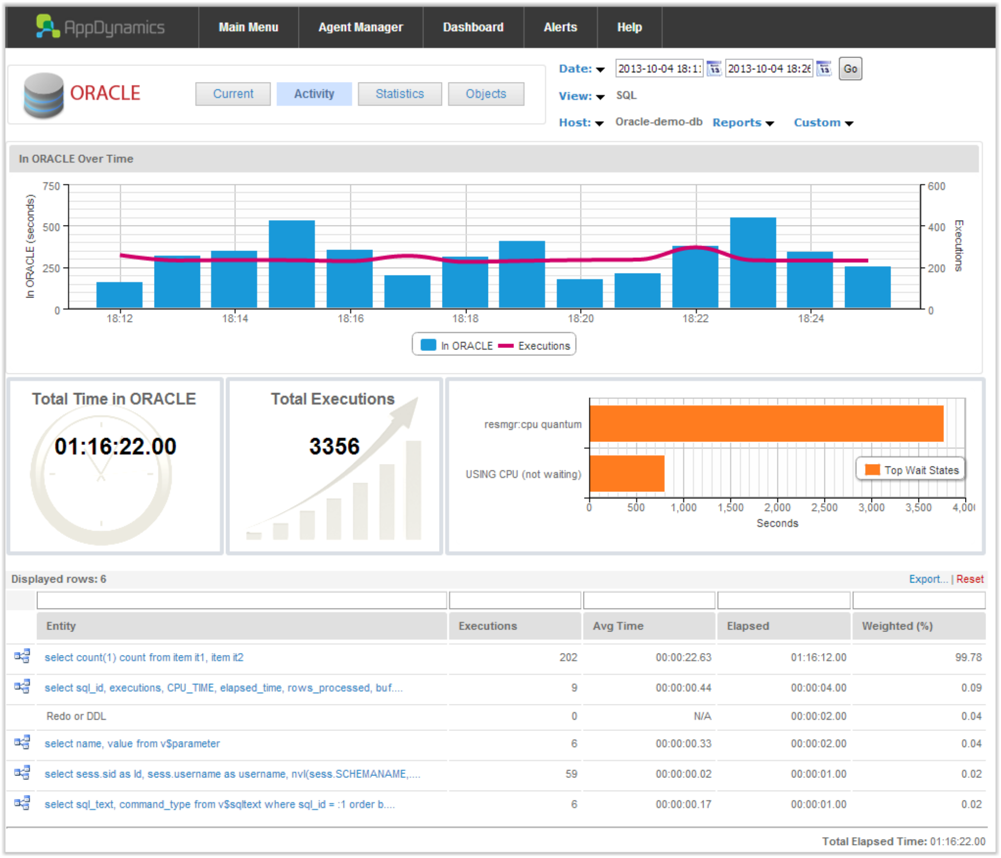
To monitor Oracle databases from the Controller

1. Right-click the Oracle database and click **Link to AppDynamics for Databases**.



> ⓘ In addition to the flow map, you can right-click the Oracle database in **Servers > Databases** to display the **Link to AppDynamics for Databases**.

AppDynamics for Databases looks for a database match in its repository and displays the database platform window.

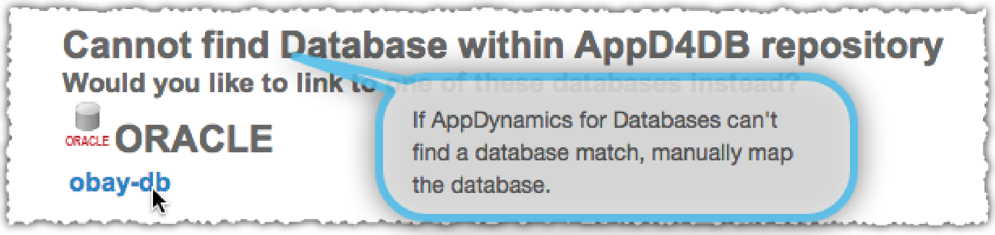


Displayed rows: 6 Export... | Reset

Entity	Executions	Avg Time	Elapsed	Weighted (%)
select count(1) count from item it1, item it2	202	00:00:22.63	01:16:12.00	99.78
select sql_id, executions, CPU_TIME, elapsed_time, rows_processed, buf....	9	00:00:00.44	00:00:04.00	0.09
Redo or DDL	0	N/A	00:00:02.00	0.04
select name, value from v$parameter	6	00:00:00.33	00:00:02.00	0.04
select sess.sid as Id, sess.username as username, nvl(sess.SCHEMANAME,....	59	00:00:00.02	00:00:01.00	0.02
select sql_text, command_type from v$sqltext where sql_id = :1 order b....	6	00:00:00.17	00:00:01.00	0.02

Total Elapsed Time: 01:16:22.00

2. If it doesn't find a match, you can select your database from a list and manually map it. You only need to map the database once. The next time you link, the database platform window displays automatically.



Cannot find Database within AppD4DB repository
Would you like to link to one of these databases instead?
ORACLE
obay-db
If AppDynamics for Databases can't find a database match, manually map the database.

For information on manually mapping your database, see Manually Map a Database.

3. See Monitor Databases for more information on monitoring with AppDynamics for Databases.

Monitor RabbitMQ Backends for .NET

On this page:

- Remote Service Detection
- Exit Points and Backend Naming
- Entry Points

Related pages:

- Configure Backend Detection for .NET
- RabbitMQ Exchanges and Exchange Types
- RabbitMQ Monitoring Extension
- App Agent Node Properties

Remote Service Detection

The AppDynamics .NET Agent automatically detects RabbitMQ backends based upon calls from instrumented tiers. RabbitMQ exit points are methods that publish or push messages to a queue. RabbitMQ entry points are methods that listen or poll for new messages in the queue.

To see RabbitMQ in the list of backends, in the left navigation pane click **Servers->Remote Services**.

Exit Points and Backend Naming

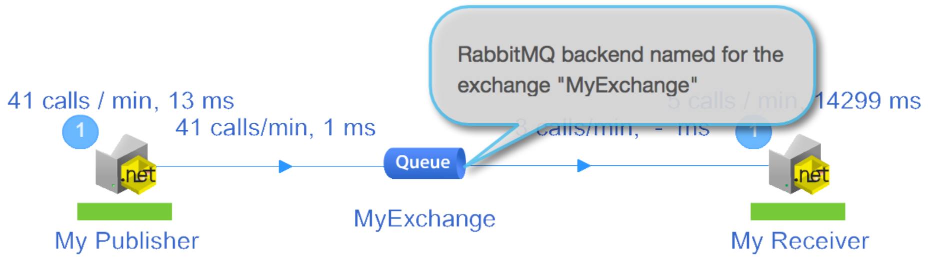
The agent discovers a RabbitMQ backend exit point when your application sends a message to the queue using the `BasicPublish()` method.

By default, the agent names the RabbitMQ backend for the exchange parameter of the `BasicPublish()` method.

For example:

```
model.BasicPublish("MyExchange", "", false, false,
    basicProperties, Encoding.UTF8.GetBytes(message));
```

In this case the agent names the queue **MyExchange**.



You can refine the backend name to include some or all segments of the routing key. To configure RabbitMQ naming you must be familiar with your implementation RabbitMQ exchanges and routing keys. See RabbitMQ Exchanges and Exchange Types.

Refine backend naming

Register the **rmqsegments** node property. For instructions on how to set a node property, see App Agent Node Properties.

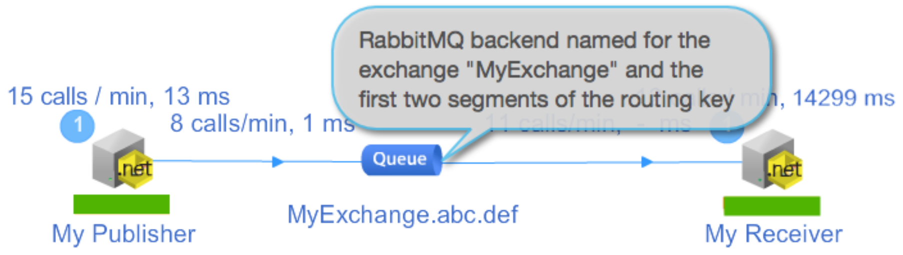
Name: rmqsegments
Description: "Configure RabbitMQ naming to include routing key segments."
Type: Integer

Value: <integer>
The routing key is a string. The agent treats dot-separated (".") substrings of the routing key as segments. Set the value to an integer that represents the number of routing key segments to include in the name.

In the following example the routing key is **abc.def.ghi**. Set the rmqsegments value to "2" to name the queue **MyExchange.abc.def**.

```
model.BasicPublish("MyExchange", "abc.def.ghi", false, false,
    basicProperties, Encoding.UTF8.GetBytes(message));
```

After you save the node property, the Controller sends the configuration to the agent. After some time the RabbitMQ backend shows up with the new name.



Entry Points

The agent discovers RabbitMQ backend entry point when your application polls the the queue. AppDynamics auto-detects RabbitMQ based upon the following patterns:

- BasicGet
- HandleBasicDeliver

BasicGet Method

The agent detects the pattern below where the application periodically polls the message queue using the `BasicGet()` method. The call timing is limited to time spent inside the `while` loop. The timer only starts when the BasicGet result returns a value at line 4. The timer ends when the next BasicGet executes. In this example, the application polls every five seconds, so the execution time equals the time in the `if` loop plus five seconds.

```
while (true)
{
        var result = channel.BasicGet("MyExchange", true);
        if (result != null)
        {
                  var body = result.Body;
                  var message = Encoding.UTF8.GetString(body);
                  Console.WriteLine("Received: {0}.", message);
        }

        Thread.Sleep(5000);
}
```

HandleBasicDeliver Method

The agent detects the `HandleBasicDeliver()` method for custom implementations of the `IBasicConsumer` interface. In this case the call timing reflects the execution time for the HandleBasicDeliver method.

Monitor Windows Hardware Resources

Related pages:

- Monitor IIS
- Monitor CLRs
- Enable Monitoring for Windows Performance Counters

The AppDynamics .NET Agent includes an embedded .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. Among other things, the Machine Agent regularly gathers system performance data metrics such as:

- CPU activity
- Memory usage
- Disk reads and writes
- Network traffic
- *New in 4.0.2*, percent free disk space and megabytes free

View Windows Hardware Resource metrics in the Metric Browser.

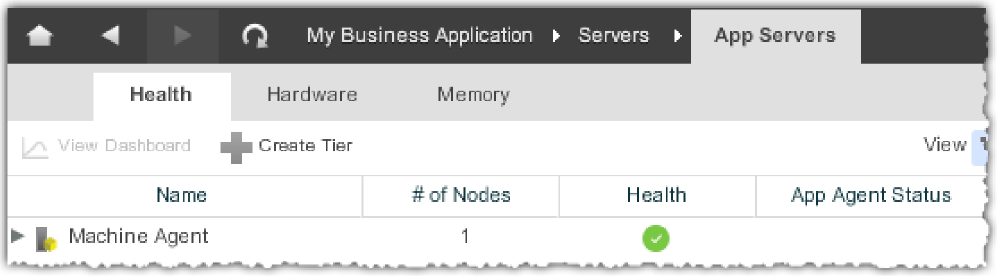
When IIS is installed on the machine, the .NET Machine Agent also reports IIS, ASP.NET and ASP.NET Application metrics. See Monitor IIS.

The .NET Machine Agent also reports CLR metrics, which are based on performance counters. See Monitor CLRs.

Machine Agent Tier

Immediately after you install and configure the .NET Agent, the .NET Machine Agent registers with the Controller and starts reporting performance data.

Frequently the machine agent metrics reach the controller before the app agent has had time to instrument and register IIS applications, Windows services, or standalone applications. If there are no application tiers, the machine agent registers as the Machine Agent tier.



Name	# of Nodes	Health	App Agent Status
Machine Agent	1	✓	

Once the app agent begins reporting metrics for configured application tiers, the .NET Machine Agent reports to the application tiers and stops sending data to the Machine Agent tier. If you don't instrument any IIS applications, Windows services, or standalone applications on the server, the .NET Machine Agent always reports to the Machine Agent tier.

If you install the Standalone Machine Agent, a Java application, in a .NET environment, then you can add custom monitors and extensions such as the HTTP listener. See Configure the .NET Machine Agent for details.

Machine Snapshots for .NET

On this page:

- Requirements
- Working With Machine Snapshots

Related pages:

- Configure Machine Snapshots for .NET
- Monitor Infrastructure
- Transaction Snapshots

Watch the video:

What is a Machine Snapshot?

Sometimes environmental conditions cause trouble on the machines where your application runs. Issues that occur outside monitored application code don't show up on transaction snapshots. Machine snapshots provide critical details about CPU usage, memory usage, and the IIS queue on a server at a specific moment in time. Use the data in machine snapshots to uncover and resolve environmental problems.

AppDynamics generates machine snapshots to capture the state of a server at a specific moment in time. The machine snapshots show the processes running on the machine, the IIS application pool activity, and any related transaction snapshots. By default the .NET Machine Agent takes machine snapshots under the following conditions:

- Periodic collection: The agent takes one snapshot every 10 minutes.
- Breached thresholds: The .NET Machine agent takes samples of machine statistics every 10

seconds within a 10 minute window. For each sample, the agent checks the CPU percent usage, the memory percent usage, and oldest item in the IIS application pool queue. The agent flags a sample as a violation when the current usage meets or exceeds one of the following thresholds:

- CPU at 80% or higher
- Memory at 80% or higher
- IIS application pool queue item older than 100 milliseconds

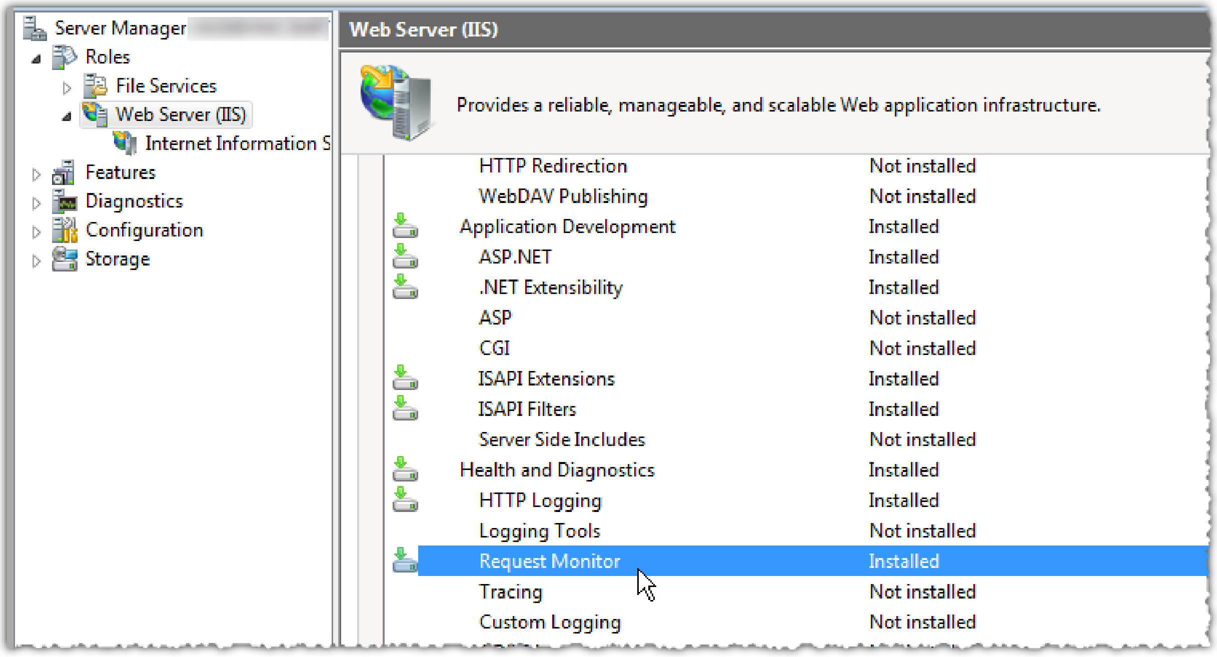
The agent takes a snapshot when it identifies 6 violations of a single type, such as CPU usage, within the window. The agent only takes one snapshot per window for breached thresholds.

To customize the periodic collection or threshold settings, see Configure Machine Snapshots for .NET.

Requirements

The .NET Machine Agent requires the following conditions to return machine snapshot data for IIS application pools:

- IIS 7 or later
- Enable the Request Monitor for the IIS Health Monitoring feature.



Provides a reliable, manageable, and scalable Web application infrastructure.

HTTP Redirection	Not installed
WebDAV Publishing	Not installed
Application Development	Installed
ASP.NET	Installed
.NET Extensibility	Installed
ASP	Not installed
CGI	Not installed
ISAPI Extensions	Installed
ISAPI Filters	Installed
Server Side Includes	Not installed
Health and Diagnostics	Installed
HTTP Logging	Installed
Logging Tools	Not installed
Request Monitor	Installed
Tracing	Not installed
Custom Logging	Not installed

✓ To enable Request Monitor from the Windows PowerShell command line, launch PowerShell as an administrator and run the following command:

```
Install-WindowsFeature Web-Request-Monitor
```

Working With Machine Snapshots

The Machine Snapshot tab on the application dashboard lists all the snapshots for the selected time range. Entries for individual snapshots show the following:

- Time the .NET Machine Agent took the snapshot
- Machine name for the snapshot
- Snapshot trigger, either periodic snapshot collection or threshold exceeded
- Percent CPU usage
- Percent memory usage

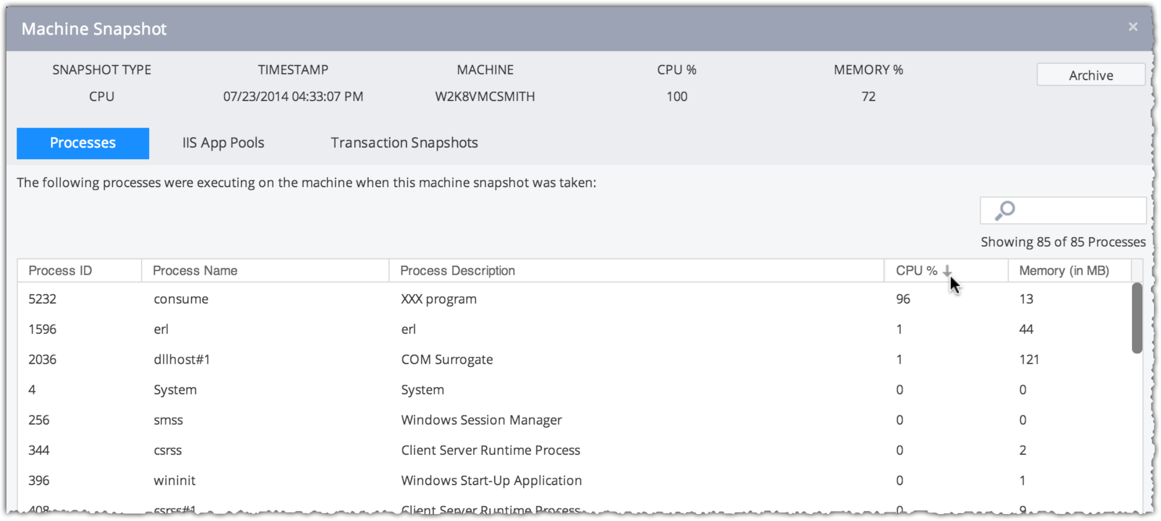
Use **Filters** to limit the snapshot list to a specific snapshot trigger:

- Periodic collection
- Memory events
- CPU events
- IIS events
- Machines where the agent took snapshots
- Archived snapshots

Double-click a snapshot to open the Machine Snapshot window:

- The Processes tab shows information similar to that from the Windows Task Manager, including the following:
    - Process id
    - Process name
    - Process description
    - Percent CPU used by the process
    - Percent memory used by the process

  Click a column heading to sort by the column. For example, click **CPU %** and sort descending to identify processes using the most CPU.



SNAPSHOT TYPE	TIMESTAMP	MACHINE	CPU %	MEMORY %
CPU	07/23/2014 04:33:07 PM	W2K8VMCSMITH	100	72

The following processes were executing on the machine when this machine snapshot was taken:

Showing 85 of 85 Processes

Process ID	Process Name	Process Description	CPU %	Memory (in MB)
5232	consume	XXX program	96	13
1596	erl	erl	1	44
2036	dllhost#1	COM Surrogate	1	121
4	System	System	0	0
256	smss	Windows Session Manager	0	0
344	csrss	Client Server Runtime Process	0	2
396	wininit	Windows Start-Up Application	0	1
408	csrss#1	Client Server Runtime Process	0	9

- The IIS App Pools tab shows information on the active IIS application pools:
    - Application pool name
    - Arrivals to the queue per second
    - Queue requests processed per second
    - Age of the oldest item in the queue

- The Transaction Snapshots tab displays all transaction snapshots involving the current machine for five minutes prior to and five minutes after the machine snapshot. This demonstrates how current environmental factors are affecting your business transaction performance. For example, when CPU usage is high, you may discover slow and stalled transactions.

  Double-click a transaction snapshot to open it.

Monitor Memory Usage with Object Instance Tracking for .NET

On this page:

Related pages:

- CLR Memory Diagnostics

When you enable object instance tracking for a node, AppDynamics analyzes the heap to identify classes with the most instances on the heap. AppDynamics tracks the top 20 .NET framework classes and the top 20 application classes based upon the number of instances. Use object instance tracking to identify memory usage trends in classes with large numbers of instances.

Requirements

- The .NET Agent only supports object instance tracking for .NET 4 and later.
- The .NET Agent Coordinator Service must be the same architecture as the monitored application. For example, a 64-bit coordinator with a 64-bit IIS application.

Working With Object Instance Tracking

Enable object instance tracking on a node

1. In the left navigation pane, click **Servers** > **App Servers** > **<tier>** > **<node>**.
2. On the node dashboard, click the Memory tab.
3. Click the Object Instance Tracking subtab.
4. Click **ON**.
   Once the agent completes the heap analysis, AppDynamics begins to track the top 20 application classes and the top 20 system (core .NET) classes in the heap.

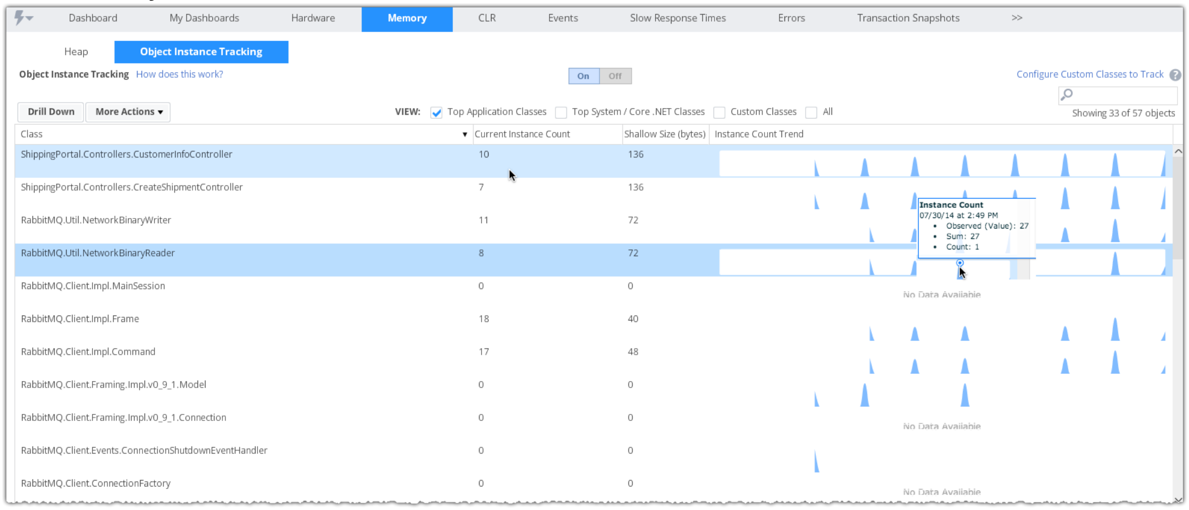
   ⚠ When you enable object instance tracking, the application pauses during heap analysis and cannot process requests. Enable object instance tracking while you're diagnosing memory issues. Turn it off when you finish troubleshooting.

Identify memory usage problems with object instance tracking

Use the following guidelines to identify memory usage problems:

- The controller resolution for object instances is one minute, but the .NET Agent agent only sends data every 10 minutes, so .NET memory appears as a series of peaks over time.

- It is normal for the controller to display 0 for the Current Instance Count and Shallow Size in between instance count collection times.
- Hover over a peak to display information about the instance count.
- Look for trends where the peaks increase in size from left to right. This may be an indication of a memory leak.



Object Instance Tracking

VIEW: [✓ Top Application Classes] Top System / Core .NET Classes Custom Classes All

Showing 33 of 57 objects

Class	Current Instance Count	Shallow Size (bytes)	Instance Count Trend
ShippingPortal.Controllers.CustomerInfoController	10	136	
ShippingPortal.Controllers.CreateShipmentController	7	136	
RabbitMQ.Util.NetworkBinaryWriter	11	72	
RabbitMQ.Util.NetworkBinaryReader	8	72	
RabbitMQ.Client.Impl.MainSession	0	0	No Data Available
RabbitMQ.Client.Impl.Frame	18	40	
RabbitMQ.Client.Impl.Command	17	48	
RabbitMQ.Client.Framing.Impl.v0_9_1.Model	0	0	
RabbitMQ.Client.Framing.Impl.v0_9_1.Connection	0	0	No Data Available
RabbitMQ.Client.Events.ConnectionShutdownEventHandler	0	0	
RabbitMQ.Client.ConnectionFactory	0	0	No Data Available

Instance Count
07/30/14 at 2:49 PM
- Observed (Value): 27
- Sum: 27
- Count: 1

Track object instances for custom classes

If you want to track a class that doesn't appear in the top 20 on the Object Instance Tracking tab, you can configure a specific class to track.

1. On the Object Instance Tracking tab, click **Configure Custom Classes to Track** to navigate to the Configure Instrumentation > Memory Monitoring tab.
2. Click **Add**.
   The Create New Instance Tracker window opens.
3. Leave **Enabled** checked.
4. Enter the fully qualified class name for the instance to track.
5. Click **Save**.

Monitor CLRs

On this page:

Related pages:

- Enable Monitoring for Windows Performance Counters

The AppDynamics .NET Agent includes the .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. Among other things, the .NET Machine Agent regularly gathers CLR performance data and reports it back to the Controller as metrics.

CLR Events

The .NET Machine Agent monitors for CLR shutdown and restart events:

- The agent reports an App Server Restart event and indicates if the restart was graceful or not.
- For non-graceful shutdown, if the agent detects a crash, it reports a CLR Crash event.

  The default exit code for a graceful shutdown is "0", if your Windows service or standalone application uses a different exit code, see "Profiler - Successful Exit Code Element" on .NET Agent Configuration Properties.

CLR Metrics

CLR metrics give insight into how the .NET runtime is performing. The AppDynamics preconfigured CLR metrics include:

- .NET CLR memory usage
- Total classes loaded and how many are currently loaded
- Garbage collection time spent, and detailed metrics about GC memory pools and caching
- Locks and thread usage
- Memory heap and non-heap usage, including the large object heap
- Percent CPU process usage

Alerting for CLR Health

You can set up health rules based on the infrastructure metrics. Once you have a health rule, you can create specific policies based on health rule violations. One type of response to a health rule violation is an alert.

In addition to the default metrics, you may be interested in additional metrics. You can specify additional performance counters to be reported by the .NET Machine Agent. See Enable Monitoring for Windows Performance Counters for details.

Once you add a custom metric you can create a health rule for it and receive alerts when conditions indicate problems.

See Alert and Respond for details on health rules and policies.

Monitor CLR Crashes

On this page:

The .NET Machine Agent monitors Windows for non-graceful CLR shutdowns on IIS, Windows services, and standalone applications. The agent raises CLR crash events in the Controller when such crashes occur. Use a policy to alert responsible parties when the agent reports a CLR crash.

Monitor for CLR Crash Events

Create a policy that sends notifications when a CLR crash event occurs.

1. Verify your **Email / SMS Configuration**. See Configure the SMTP Server.
2. Create a notification action configured to alert the parties who respond to CLR crashes. See Notification Actions.
3. Create a policy to trigger on **CLR Crash** under **Other Events** on the Create Policy window. For the action, choose the notification action you created in step 2. See Configure Policies.

After you create the policy, the Controller sends a notification when the .NET Machine Agent raises CLR crash events.

Analyze and Respond to CLR Crashes

When a CLR crashes, the .NET Machine Agent raises an event in the Controller. The events pane on the application dashboard displays the number of server crash events, including CLR crashes, during the selected time range.

1. Click the **Server Crashes** link in the application dashboard to display a list of all server crash events during the selected time range.
2. Optionally filter on the **CLR Crash** event type.

   Entries for individual CLR crashes display information about the crash event, including the process id and the Windows event log id.
   After a CLR crash, IIS automatically tries to restart the CLR, so one issue frequently causes multiple CLR crash events.

3. Double-click a CLR Crash type event to display more information in the **CLR Crash** window:
   - The **Summary** tab shows the Windows process id of the w3wp process that crashed and the Windows event log id for the crash. The summary also includes the affected tier and node names.
     If multiple tiers have nodes on the same machine, you may see more than one tier in the CLR Crash window.
   - The **Details** tab displays essential information about the crash.
   - The **Actions Executed** tab shows any actions the event triggers. See Monitor for CLR crash events.
   - Read and add new comments on the **Comments** tab. For example, add a comment to the CLR crash event to notify colleagues that you're triaging the issue.

Disable CLR Crash Event Reporting

The .NET Machine Agent enables CLR crash event reporting by default. Disable it as follows:

1. Edit the agent config.xml as an administrator. See Where to Configure App Agent Properties.
2. Set **enabled="false"** for the CLR Crash Reporting element. See CLR Crash Reporting Element.

3. Restart the AppDynamics.Agent.Coordinator service.

Crash Events the .NET Machine Agent Monitors

The .NET Machine Agent listens on the following Windows event logs for crash events:

- Application Log
- System Log

The agent listens for events logged at event level "Warning" and higher on the following sources:

- Application Error
- .NET Runtime
- WAS (Windows Process Activation Service)

The agent listens for events logged at event level "Information" and higher on the following source:

- Windows Error Reporting

The agent reports CLR crashes as events to the Controller as follows:

- all w3wp process crashes
- instrumented Windows service crashes
- instrumented standalone applicationcrashes

To monitor CLR crashes only for instrumented w3wp processes, see "Process Monitor Element" on .NET Agent Configuration Properties.

The Controller treats all events as "Warning" severity, which is not directly related to the Windows event level.

Monitor IIS

On this page:

Related Pages:

- Monitor CLR Crashes

The AppDynamics .NET Agent includes the .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. Among other things, the .NET Machine Agent regularly gathers IIS performance data and reports it back to the Controller as metrics.

IIS Events

The .NET Machine Agent monitors IIS for shutdown and restart events:

- The agent reports an App Server Restart event and indicates if the restart was graceful or not.
- For non-graceful shutdown, if the agent detects a crash, it reports a CLR Crash event.

Default IIS Metrics for .NET

The .NET Machine Agent uses Microsoft Windows Performance Counters to gather IIS metrics. In the Controller, you can view preconfigured metrics for IIS in the Metric Browser.

Internet Information Services (IIS) must be installed on the machine for you to view the metrics for the following:

IIS Metrics

From the Metric Browser:

- To view the IIS metrics for a tier, expand **Application Infrastructure Performance -> <Tier> -> IIS**.
- To View the IIS metrics for a node, expand **Application Infrastructure Performance -> <Tier> -> Individual Nodes -> <Node> -> IIS**.

AppDynamics reports each metric for the entire tier, each individual application pool, and each individual node as follows:

- **Application Infrastructure Performance -> <Tier> -> IIS** = combined for all IIS processes in all Application Pools for this tier
- **Application Infrastructure Performance -> <Tier> -> Application Pools -> <application pool name>** = combined for all processes in this specific Application Pool
- **Application Infrastructure Performance -> <Tier> -> Individual Nodes -> <Node>** = metrics for the specific node

ASP.NET Metrics

To view the ASP.NET metrics in the Metric Browser, expand **Application Infrastructure Performance -> <Node> -> ASP.NET**.

AppDynamics reports the following ASP.NET metrics:

- Application Restarts
- Applications Running
- Requests Disconnected
- Requests Queued
- Requests Rejected
- Request Wait Time
- Worker Process Restarts

ASP.NET Application Metrics

To view the ASP.NET Application metrics in the Metric Browser, expand **Application Infrastructure Performance -> <Node> -> ASP.NET Applications**.

AppDynamics reports the following ASP.NET Application metrics:

- Anonymous Requests
- Anonymous Requests/sec
- Cache Total Entries
- Cache Total Hit Ratio
- Cache Total Turnover Rate
- Cache API Entries
- Cache API Hit Ratio
- Cache API Turnover Rate
- Errors Unhandled During Execution/sec
- Errors Total/sec
- Errors During Preprocessing
- Errors During Compilation
- Errors During Execution
- Errors Unhandled During Execution
- Errors Unhandled During Execution/sec
- Errors Total
- Errors Total/sec
- Output Cache Entries
- Output Cache Hit Ratio
- Output Cache Turnover Rate
- Pipeline Instance Count
- Requests Executing
- Requests Failed
- Requests In Application Queue
- Requests Not Found
- Requests Not Authorized
- Requests Succeeded
- Requests Timed Out
- Requests Total
- Requests/sec
- Session State Server Connections Total
- Session SQL Server Connections Total
- Sessions Active
- Sessions Abandoned
- Sessions Timed Out
- Sessions Total
- Transactions Aborted
- Transactions Committed
- Transactions Pending
- Transactions Total
- Transactions/sec

Monitor IIS Application Pools

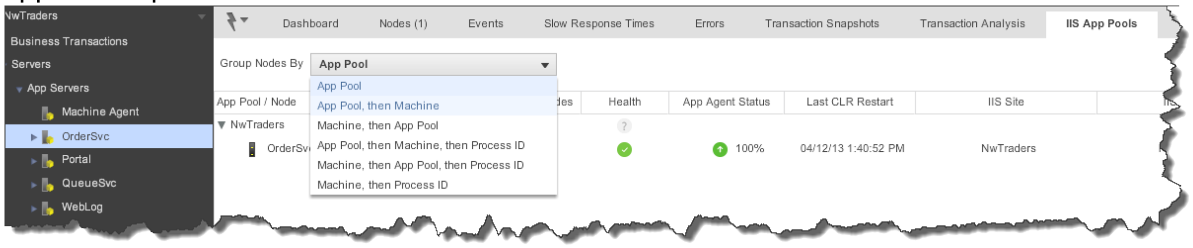
You can monitor the health of IIS application pools for the instrumented .NET nodes in a tier. You can view the information by application pool, machine, and process IDs in varying hierarchies.

These groupings enable you to visualize key performance indicators for your infrastructure:

- Health of the node in the specified time-range for a particular group.
- App Server Agent's status in the specified time-range.
- Last CLR restart.
- A link to the parent tier.

View the IIS application pools:

1. In the left navigation pane, select the tier that you want to monitor: **Servers -> AppServers -> <Tier>**.
2. Click the **IIS App Pools** tab.
3. (Optional) From the **Group Nodes By** dropdown menu, select how you want to view the application pools and machines for this tier.



4. To view a node's dashboard, double-click the node in the list. From there you can select the various tabs for details about performance of the node.
   If a machine or application pool name is not available for a .NET node, an "**Unknown App Pool**" / "**Unknown Machine**" grouping is created.

Troubleshoot Slow Response Times for .NET

On this page:

- Initial Troubleshooting Steps
- .NET Resource Troubleshooting
- Need More Help?

Related pages:

- Monitor Windows Hardware Resources

You can learn that your application's response time is slow in the following ways:

- You receive an alert: If you have received an email alert from AppDynamics that was configured through the use of health rules and policies, the email message provides a number of details about the problem that triggered the alert. See Email Notifications. If the problem is related to slow response time, see Initial Troubleshooting Steps.
- You view the Application Dashboard for a business application and see slow response times.
- A user reported slow response time that relates to a particular business transaction, for

example, an internal tester reports "Searching for a hotel is slow".

Initial Troubleshooting Steps

In some cases, the source of your problem might be easily diagnosed by choosing **Troubleshoot -> Slow Response Times** in the left navigation pane. See Troubleshoot Slow Response Times.

.NET Resource Troubleshooting

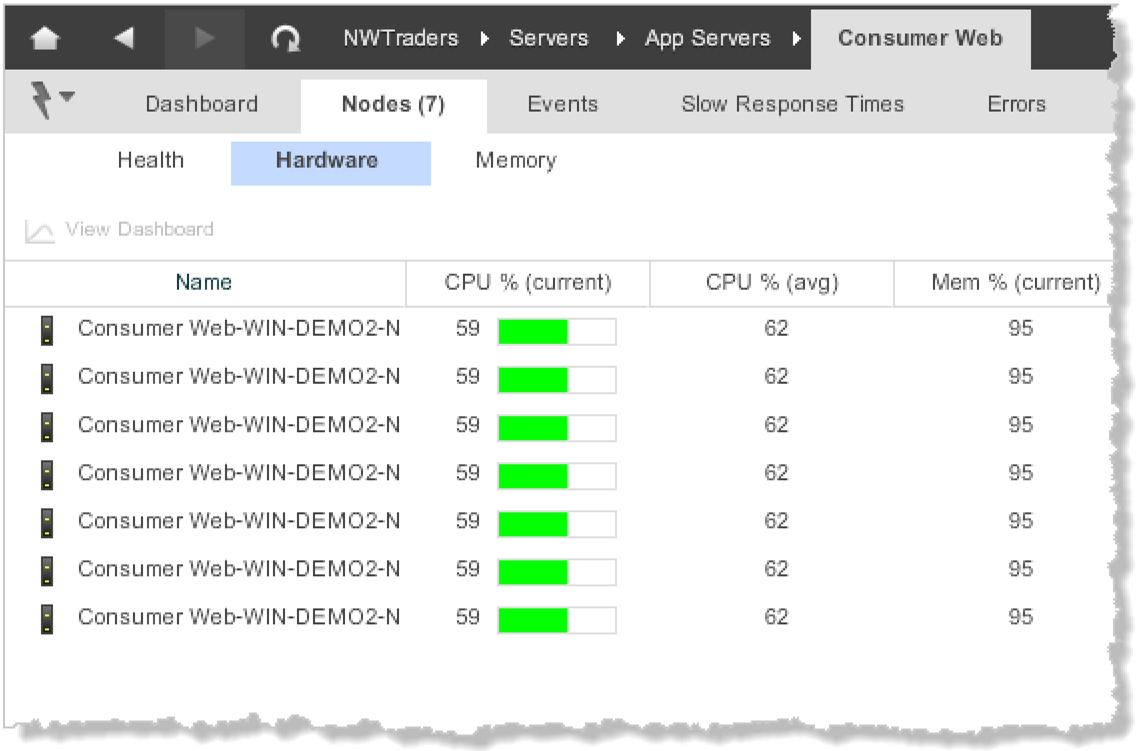
If you've tried to diagnose the problem using those techniques and haven't found the problem, use the following troubleshooting approaches to find other ways to determine the root cause of the issue.

Step 1 - CPU saturated?

Is the CPU of the CLR saturated?

How do I know?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then click the Hardware tab.
3. Sort by CPU % (current).



Name	CPU % (current)	CPU % (avg)	Mem % (current)
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95
Consumer Web-WIN-DEMO2-N	59	62	95

If the CPU % is 90 or higher, the answer to the question in Step 4 is Yes. Otherwise, the answer is No.

Yes – Go to Step 2

No – Review various metrics in the Metric Browser to pinpoint the problem.

In the left navigation pane, click **Servers -> App Servers -> <slow tier>**. Review these metrics in particular:

- ASP.NET -> Application Restarts
- ASP.NET -> Request Wait Time
- ASP.NET -> Requests Queued

- CLR -> Locks and Threads -> Current Logical Threads
- CLR -> Locks and Threads -> Current Physical Threads

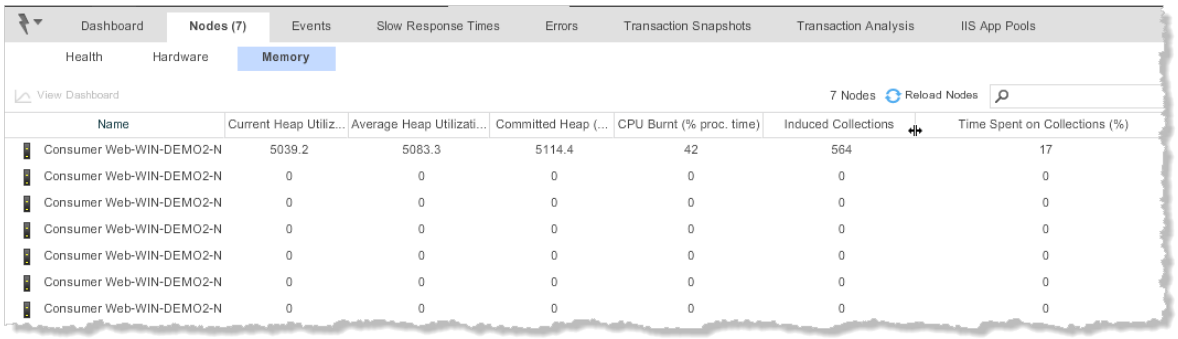
- IIS -> Number of working processes
- IIS -> Application pools -> <Business application name> -> CPU%
- IIS -> Application pools -> <Business application name> -> Number of working processes
- IIS -> Application pools -> <Business application name> -> Working Set

You have isolated the problem and don't need to continue with the rest of the steps below.

Step 2 - Significant garbage collection activity?

How do I know?

1. Display the Tier Flow Map.
2. Click the Nodes tab, and then click the Memory tab.
3. Sort by Time Spent on Collections (%) to see what percentage of processing time is being taken up with garbage collection activity.



Name	Current Heap Utiliz...	Average Heap Utilizati...	Committed Heap (...	CPU Burnt (% proc. time)	Induced Collections	Time Spent on Collections (%)
Consumer Web-WIN-DEMO2-N	5039.2	5083.3	5114.4	42	564	17
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0
Consumer Web-WIN-DEMO2-N	0	0	0	0	0	0

If Time Spent on Collections (%) is higher than acceptable (say, over 40%), the answer to the question in Step 5 is Yes. Otherwise, the answer is No.

Is there significant garbage collection activity?

Yes – Go to Step 3.

No – Use your standard tools to produce memory dumps; review these to locate the source of the problem.

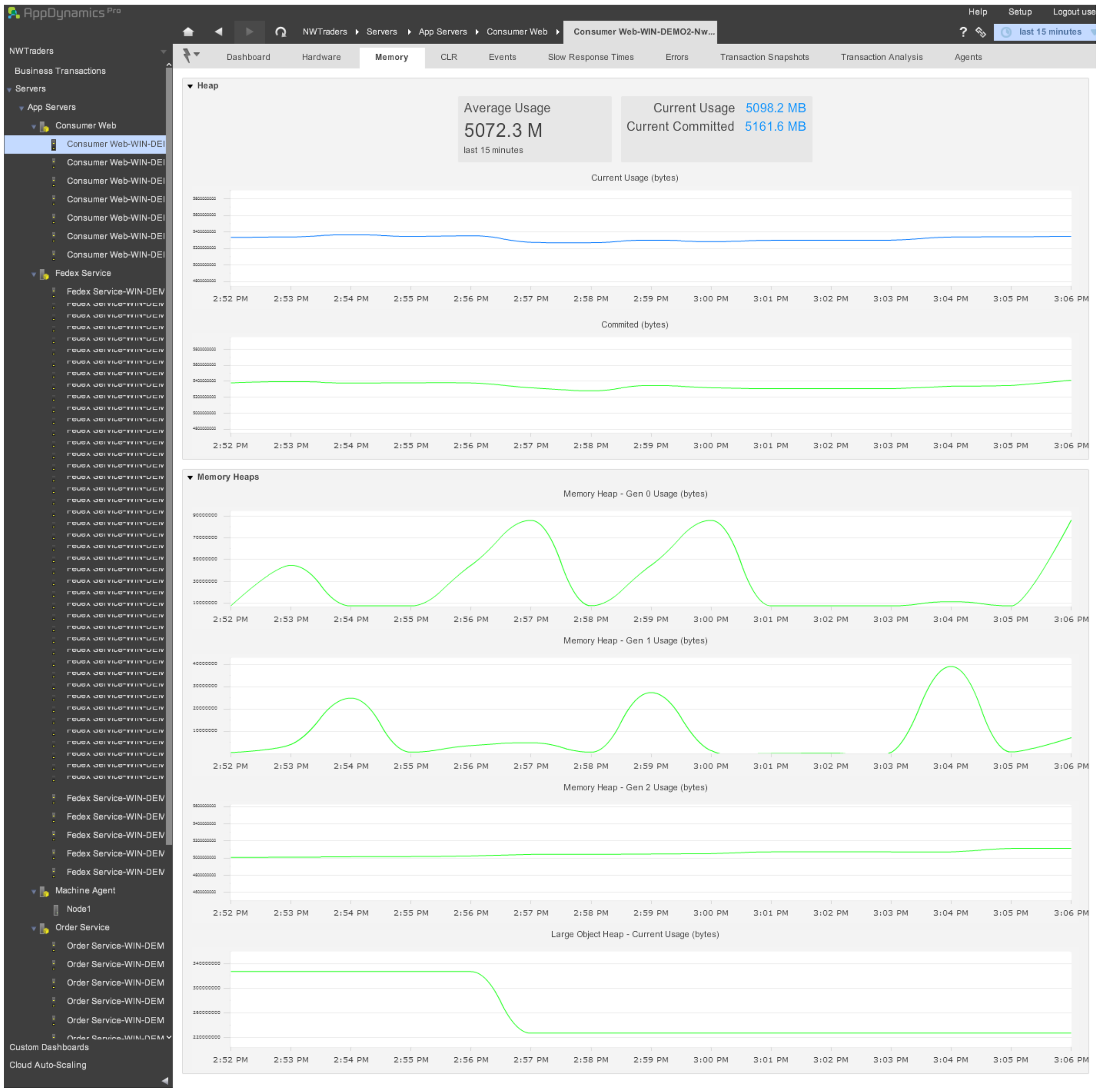
You have isolated the problem and don't need to continue with the rest of the steps below.

Step 3 - Memory leak?

Is there a memory leak?

How do I know?

1. From the list of nodes displayed in the previous step (when you were checking for garbage collecting activity), double-click a node that is experiencing significant GC activity.
2. Click the Memory tab, then review the committed bytes counter and the size of the Gen0, Gen1, Gen2 and large heaps.



If memory is not being released (one or more of the above indicators is trending upward), the answer to the question in Step 6 is Yes. Otherwise, the answer is No.

Yes – Use your standard tools for troubleshooting memory problems. You can also review ASP.NET metrics; click **Servers -> App Servers -> <slow tier> -> ASP.NET**.

No – Use your standard tools to produce memory dumps; review these to locate the source of the problem.

Whether you answered Yes or No, you have isolated the problem and don't need to continue with the rest of the steps below.

Need More Help?

If slow response time persists even after you've completed the steps outlined above, you may need to perform deeper diagnostics. If you can't find the information you need on how to do so in the AppDynamics documentation, consider posting a note about your problem in a community discussion topic. These discussions are monitored by customers, partners, and AppDynamics staff. Of course, you can also contact AppDynamics support.

- Community Discussion Boards (If you don't see AppDynamics Pro as a topic, click Sign In at the upper right corner of the screen.)