Monitoring WildFly JBoss Server

eG Innovations Product Documentation

www.eginnovations.com

Total Performance Visibility

Table of Contents

Table of Figures

Chapter 1: Monitoring WildFly JBoss servers

WildFly is a flexible, lightweight, managed application runtime that helps you build various applications such as Java applications, web applications, and portals, and also offers extended enterprise services such as clustering, caching, and persistence.

WildFly offers two modes: a traditional, single JVM, standalone mode, and a multi-JVM option, domain mode, which synchronizes configuration across any number of processes and hosts. The domain mode adds a central control point, the domain controller, for all of your systems, though the management capabilities of both the standalone and domain modes are the same.

Once the WildFly JBoss server is deployed in either standlaone mode or the domain mode, the eG agent periodically executes tests on the WildFly JBoss server, collects the necessary statistics, and reports them to the eG manager. These tests are mapped to specific layers of the WildFly JBoss server's layer model (see Figure 1.1).

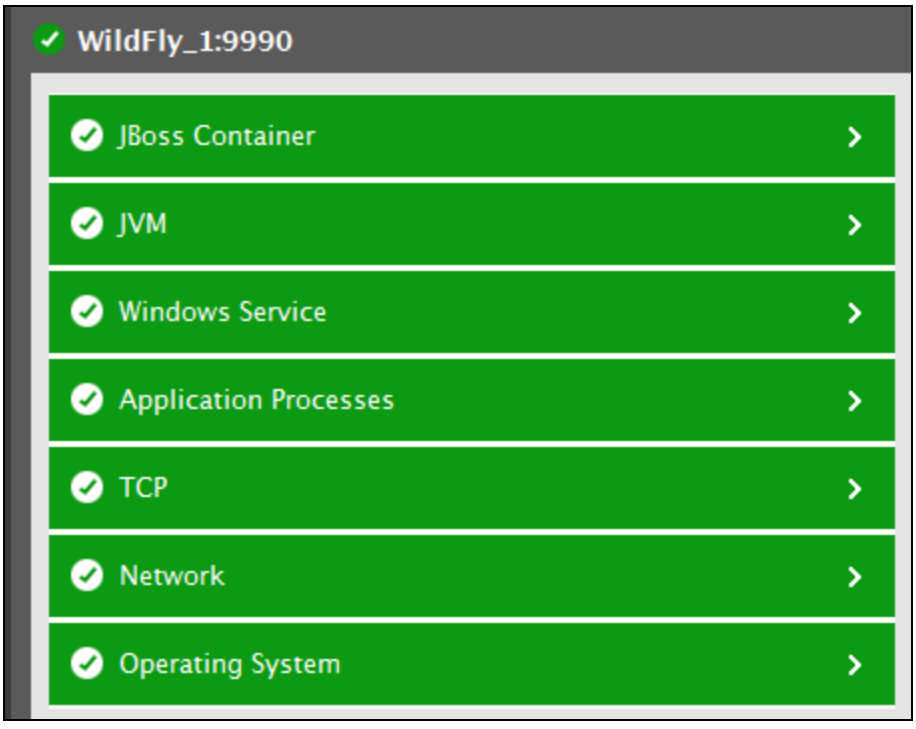

Figure 1.1: The layer model of the WildFly JBoss server

Using the metrics reported, administrators can find quick and accurate answers for the following performance questions:

- How well each connector is processing the incoming requests?
- Are enough connections available in the connection pool of the datasources and XA-datasources, or do more connections have to be allotted to the pool?
- Are there adequate prepared statements available? How well the prepared statements are accessed from the prepared statement cache?
- How frequently was the servlet invoked?
- Does the servlet take too long to execute?
- How frequently are EJBs created/removed?
- Are there any EJBs that are ready to service clients?
- How many EJBs are currently in the EJB pool?
- Are too many messages have been enqueued?
- How quickly does the queue process messages within?
- Does the topic take too long to process messages?
- What type of messages are currently available in the topic - durable or non-durable?
- How many durable subscribers are there to a topic?
- How well each type of transaction is processed?

The **Operating System**, **Network**, **TCP**, **Application Processes** and **Windows Services** layers of the WildFly JBoss monitoring model is similar to that of a Windows Generic server model. Refer to the *Monitoring Unix and Windows Servers* document to know more about the tests pertaining to these layers. The JVM layer of this server is similar to that of the Java Application server monitoring model. Since the tests pertaining to these layers have been dealt with in the *Monitoring Java Application servers* document, The JBoss Container Layer focuses on the **JBoss Server** layer.

1.1 Creating a new management user in the WildFly JBoss server

The management interfaces in a WilFly JBoss server are secured by default, and hence there is no default user. This is a security precaution, to prevent security breaches from remote systems due to simple configuration errors. Without a user, administrators may not be able to use the web-based Management Console of the WildFly JBoss server. It is therefore mandatory to create an initial administrative user, who will be able to use the web-based Management Console and remote instances of the Management CLI to configure and administer the WildFly JBoss serrver from remote systems. This user can be either the **Management user** or the **Application user**. A **Management user** is added to the **ManagementRealm** of the WildFly Jboss and is authorized to perform management operations using the

web-based management console or the Management CLI. On the other hand, the **Application user** is added to the **ApplicationRealm** and this user has no particular permissions and is provided for use with applications. In order to monior the WildFly JBoss server, a user has to be created so that the WildFly JBoss server is accessed through the web-based management console. Therefore it is necessary to add a **Management user** in the WildFly JBoss server. Let us now discuss the steps on how to add a management user below:

1. In order to add a user to the WildFly JBoss server, you will require either one of the following files available in the **<JBOSS_INSTALL_DIR\bin>** location:
   - **add-user.sh**
   - **add-user.bat**
2. Execute the **add-user.bat** file in case the WildFly JBoss server is installed on a Windows environment and execute the **add-user.sh** file in case the WildFly JBoss server is installed on a Linux environment.
3. Once the file is executed, you will be required to choose the type of the user that you wish to add. If you wish to add a **Management User** specify **a** or if you wish to add an **Application User**, specify **b** as mentioned below.

```
What type of user do u wish to add?
Management User (mgmt-user.properties)
Application User (application-users.properties)
(a): a
```

4. Specify the credentials of the user that you wish to add. In our example, let us consider the **Username** as 'elvis'. Specify the **Username** and **Password** accordingly and proceed to step 5.

```
Username: elvis
Password:*****
Re-enter Password:****
```

5. If you wish to associate the user to a group in the WildFly JBoss server installation, then you can provide a comma-separated list of groups or if you do not wish to associate the user to any group, you can do so as shown below:

```
What groups do you want this user to belong to? (Please enter a comma separated list,
or leave blanck for none):
```

6. By default, if the **Management User** option is chosen, then the user will be added to the **ManagementRealm** of the WildFly JBoss server. Specify **yes** to confirm the same so that the user is added to the **ManagementRealm**.

```
About to add user 'elvis' for realm 'ManagementRealm'
Is this correct yes/no? yes
```

7. Now, the user will be added to the **mgmt-users.properties** and the **mgmt-groups.properties** of the WildFly JBoss server installation. Then, specify **yes** if a user of any other WildFly JBoss instance needs to be authenticated to join the cluster as a member. This authentication is a mandatory requirement for a WildFly JBoss cluster setup.

```
Is this new server going to be used for one AS process to connect to another AS
process? e.g. for a slave host controller connecting to the master or for a Remoting
connection for a server to server EJB calls. yes/no? yes
```

8. Once you specify **yes** in Step 7, a secret value will appear which needs to be copied and stored separately for future reference. Whenever a new WildFly JBoss instance is added in a domain, specifying the secret value while configuring the new instance will let the new instance be the slave of the WildFly JBoss installation in a cluster setup i.e., a user will be allowed to communicate with all the associated instances once the secret value is shared.

```
To represent the user add the server-identities definition (secret
value="AWEStanW4cmziQ").
```

Once the **Management User** is created, administrators can login to the WildFly JBoss management console and perform management operations extensively.

1.1.1 Identifying the host name and server instance name of the WildFly JBoss server running in Domain mode

In order to identify the host name of the WildFly JBoss instance that is to be monitored, do the following:

If the WildFly JBoss instance to be monitored is set to operate in **Domain** mode, then, by clicking the **RunTime** tab, you can identify whether the host is configured as the master or slave in a WildFly JBoss instance cluster.

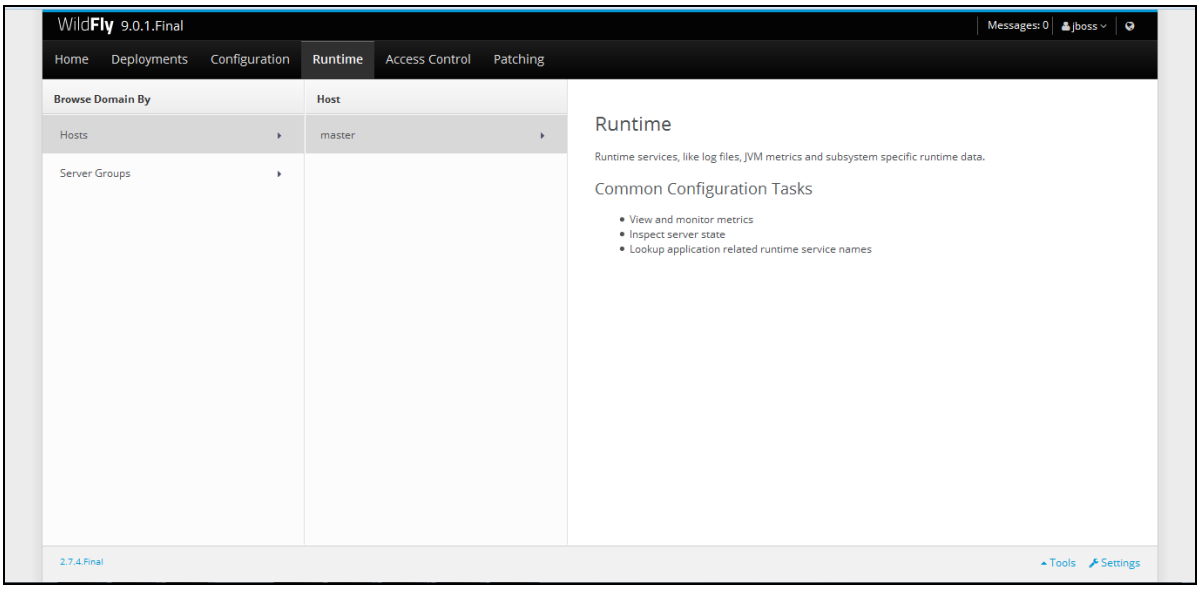

Figure 1.2: Figuring out the host name of the WildFly JBoss instance to be monitored

Uopn clicking the **Host**, the groups to which the host is associated and the server instances that are available for that host can be identified.

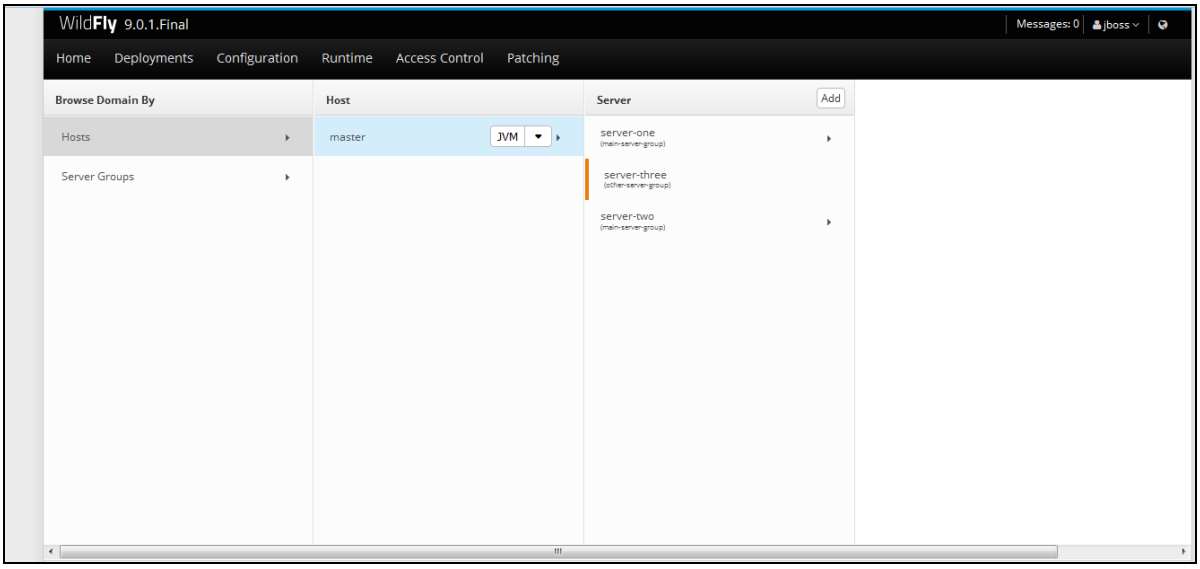

Figure 1.3: Viewing the server instances associated with the host

Specify the name of the server instance that is to be monitored in the **JBOSS SERVER INSTANCE NAME** text box when you configure the tests for the WildFly JBoss server.

If the WildFly JBoss instance to be monitored is a standalone server, the name of the server can be viewed in the **Runtime** tab page.

1.2 The JBoss Container Layer

This layer helps the administrators find quick and accurate performance measures for the following:

- The number of incoming requests processed by each connector;
- The number of connections available in the connection pool of the datasources and XA-datasources, the utilization of the connections by the datasource and the XA datatsource;
- The availability of the prepared statements; the number of prepared statements accessed from the prepared statement cache of the datasource and the XA datasource?
- The number of times the servlet was invoked;
- The time duration taken by the servlet for execution;
- The number of EJBs that are ready to service clients;
- The number of EJBs currently in the EJB pool;
- The number of messages that are enqueued in the queue;
- The time taken by the queue to process messages;
- The time taken by the topic to process the messages;
- The type of message are currently available in the topic;
- The number of durable subscribers to a topic;
- The number of transactions of various types that were processed; etc

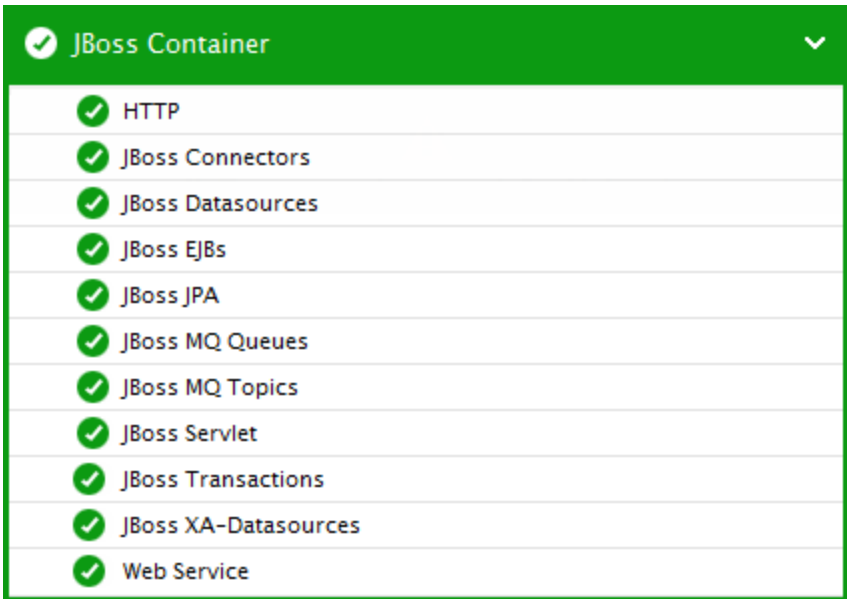

Figure 1.4: The tests mapped to the JBoss Container layer

1.2.1 JBoss Connectors Test

In a typical WildFly JBoss architecture, there are two main web connectors namely the Java I/O Connector and the AJP Connector. While the Java I/O connector uses the Java I/O to serve HTTP/HTTPS connections directly to the platform, the AJP Connector uses Apache's Portable Runtime (APR) native code library. Often, the connectors of the WildFly JBoss server receive a large number of requests for processing. In such cases, it becomes inevitable to identify how many requests are processed by the connector, how long it takes to process a request etc. The **JBoss Connectors** test exactly helps you figure out your concerns! This test not only monitors the number of incoming requests processed by each connector but also evaluates the time taken for processing the requests and the maximum time taken to process a single request. This test also sheds light on the amount of data used in processing the requests and errors encountered while processing the requests.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each connector on the target WildFly JBoss that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.

7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Request count:	Indicates the number of incoming requests processed by this connector.	Number	A high value is desired for this measure.
Processing time:	Indicates the total time taken by this connector to process the incoming requests.	Mins	A low value is desired for this measure. The **maxThreads** parameter of the WildFly JBoss is set to 200 by default. This implies that the **maxThreads** parameter actually creates a thread pool behind the connector and processes the incoming requests. If no threads are available or the maximum limit for the number of threads is reached, processing of new requests is delayed which eventually leads to the piling up of requests. Also, a request may take too long to be processed if the request is malign or if adequate resources are not available for processing.
Max time:	Indicates the maximum time taken by this connector to process an	Mins	An abnormally high value clearly indicates a performance issue or lack of threads that process the

Measurement	Description	Measurement Unit	Interpretation
	incoming request.		requests.
Bytes sent:	Indicates the number of bytes sent by this connector for processing the requests.	Number	Comparing the value of these measures across the connectors will help you identify the connector that is using the maximum bytes for processing the requests.
Bytes received:	Indicates the number of bytes received by this connector for processing the requests.	Number	
Error count:	Indicates the number of errors encountered by this connector while processing the requests.	Number	Ideally, the value of this measure should be zero.

1.2.2 JBoss Datasources Test

A **Datasource** is a Java Naming and Directory Interface (JNDI) object used to obtain a connection from a connection pool to a database. When connecting to a data source, JBoss Enterprise Application Platform must allocate resources and de-allocate resources for every connection. This is quite expensive in terms of time and system resources. Connection pooling reduces the cost of data source connections by creating a number ("pool") of data source connections available to be shared by applications. Pooling data source connections is much more efficient than allocating and de-allocating resources on demand. Often, administrators may find it tedious to analyze how well connections from a connection pool have been utilized by the datasource and how efficiently the datasource has been channelizing the connections. This is where the **JBoss Datasource** Test helps! This test monitors the datasources and reports the following:

- The number of connections that are currently available and the number of active connections;
- The number of connections that were created and destroyed using this datasource;
- The time taken to create a connection on this datasource and the maximum time taken for creating a connection;
- How many connections are available in a connection pool and how well the connections of a connection pool are utilized?
- The prepared statement cache size and the number of prepared statements added

- How many times prepared statements are accessed and how well the prepared statement cache caters to the requests for accessing the prepared statements?

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each datasource of the target WildFly JBoss that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Active count:	Indicates the number of active connections through this datasource.	Number	A high value is desired for this measure.
Available count:	Indicates the number of connections that are currently available for use in this datasource.	Number	A low value for this measure indicates that adequate connections are not available in the datasource.
Average blocking time:	Indicates the average time spent blocking a connection during the last measurement period.	Secs	
Average creation time:	Indicates the average time spent for creating a physical connection during the last measurement period.	Secs	Comparing the value of this measure across the datasources will reveal the datasource that is the slowest when creating a physical connection.
Created count:	Indicates the number of connections that were created using this datasource.	Number	
Destroyed count:	Indicates the number of	Number	

Measurement	Description	Measurement Unit	Interpretation
	connections that were destroyed in this datasource since the start of the server.		
Max creation time:	Indicates the maximum time taken to create a connection in this datasource.	Secs	Compare the value across the datasources to identify the datasource that is taking too long to create a connection i.e., you can identify the datasource that is the slowest.
Max used count:	Indicates the maximum number of connections that were used in this datasource.	Number	
Max wait time:	Indicates the maximum time a user had to wait for a connection through this datasource.	Secs	A low value is desired for this measure. An abnormally high value or a sudden increase in the value of this measure is a cause of concern which requires further investigation.
Timed out:	Indicates the time taken for a connection to time out.	Mins	Normally, a connection would time out if it has been waiting in the queue for a longer time.
Total blocking time:	Indicates the total time taken for a connection to fail or time out.	Secs	
Total creation time:	Indicates the time taken to create a connection through this datasource.	Secs	A low value is desired for this measure.
Maximum pool size:	Indicates the	Number	The `max-pool-size` parameter

Measurement	Description	Measurement Unit	Interpretation
	maximum number of connections that can be handled by the connection pool of this datasource.		defines the maximum size of the connection pool. This parameter is more important because it limits the number of active connections to the datasource and thus the concurrent activity on the data source. If this value is set too low it's likely that the platform's hardware resources will be underutilized.
Minimum pool size:	Indicates the minimum number of connections that the connection pool of this datasource can contain.	Number	The `min-pool-size` data source parameter defines the minimum size of the connection pool. The default minimum is zero connections, so if a minimum pool size is not specified, no connections will be created in the connection pool when the platform starts. As data source transactions occur, connections will be requested but because the pool defaults to zero on start up, none will be available. The connection pool examines the minimum and maximum parameters, creates connections and places them in the pool. Users of any affected application will experience a delay while this occurs. During periods of inactivity the connection pool will shrink, possibly to the minimum value, and later when transactions occur application performance will again suffer.
PreparedStatement cache size:	Indicates the number of prepared statements per connection to be	Number	When using the JPA annotations for queries, the result is prepared statements that will be executed against the database. Prepared

Measurement	Description	Measurement Unit	Interpretation

	kept open and reused in subsequent requests.		statements have two phases for execution: preparation of the statement, and execution of the statement. Statement preparation involves significant CPU load so to improve throughput prepared statements can be cached. Statements can be cached either via the JDBC driver or configuration of the data source.
			
			To enable caching of prepared statement, add the following two lines to the data source configuration file, a file with the pattern *-ds.xml (where the * is usually your database, such as oracle, mysql, db2, etc.) in the directory: `JBOSS_EAP_DIST/jboss-as/server/PROFILE/deploy` . Note that the minimal configuration does not support data sources.
			
			`<prepared-statement-cache-size>100</prepared-statement-cache-size>`
			
			`<shared-prepared-statements>true</shared-prepared-statements>`
			
			The first line enables the prepared statement cache and sets the size of the cache. This should be large enough to hold all the prepared statements across any and all deployed applications using this particular data source (multiple data

Measurement	Description	Measurement Unit	Interpretation
			sources can be configured). The second line states that if two requests are executed in the same transaction the prepared statement cache should return the same statement. This will avoid the use of CPU cycles in preparing the same statements over and over again, increasing throughput and decreasing response times. The actual improvement will vary according to specific circumstances but is well worth the effort.
PreparedStatement cache access count:	Indicates the number of times the prepared statement cache was accessed.	Number	
PreparedStatement cache add count:	Indicates the number of new prepared statements added i.e., cached in the prepared statement cache.	Number	
PreparedStatement cache current size:	Indicates the number of prepared statements that are currently available in the cache.	Number	If the value of this measure is close to the *PreparedStatement cache* size measure, then it indicates that the prepared statement cache is almost running out of space.
PreparedStatement cache delete count:	Indicates the number of prepared statements that are deleted from the prepared statement	Number	

Measurement	Description	Measurement Unit	Interpretation
	cache.		
PreparedStatement cache hit count:	Indicates the number of times the statement was accessed from the prepared statement cache.	Number	A high value is desired for this measure.
PreparedStatement cache miss count:	Indicates the number of times a request for a statement was not serviced from the prepared statement cache.	Number	A high value for this measure is a cause of concern.

1.2.3 JBoss EJBs Test

A bean is usually an application class that contains business logic. It may be called directly from Java code, or it may be invoked via the Unified EL i.e., Unified Expression language. A bean may access transactional resources. Dependencies between beans are managed automatically by the container. The lifecycle of a bean is always managed by the container. The beans are classified as follows:

- stateless session beans
- stateful session beans
- message driven beans, and
- entity beans

This test auto-discovers the EJBs deployed on the JBoss server and reports critical measures relating to each of the EJB in detail. Using this test, you can figure out the number of times each bean has been invoked and the instances of each bean that can be accommodated in the pool. This way, you could figure out the effective utilization of the pool and the bean in particular.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each EJB of the target WildFly JBoss that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Number of invocations:	Indicates the number of times this bean has been invoked since the start of the server.	Number	
Average execution time:	Indicates the average time taken to execute the method of this bean	Secs	A higher value for this measure is a cause for concern.

Measurement	Description	Measurement Unit	Interpretation
	during the last measurement period.		
Peak concurrent invocations:	Indicates the maximum number of times the method of this bean has been invoked.	Number	
Pool available count:	Indicates the number of instances of this bean that is currently available for use in the pool.	Number	A high value is desired for this measure.
Pool creation count:	Indicates the number of bean instances that were created in the pool.	Number	
Pool remove count:	Indicates the number of bean instances that were removed from the pool.	Number	
Pool maximum size:	Indicates the maximum number of instances of this bean that can be accommodated in the pool.	Number	
Current pool size:	Indicates the number of instances of this bean that are currently utilized in the pool.	Number	If the value of this measure is equal to the *Pool maximum size* measure, then you can increase the *Pool maximum size* measure to a more desirable value so that the bean instances are readily available for use.
Pool utilization:	Indicates the percent of beans that have been utilized from the pool.	Number	A high value is indicative of maximum utilization of the bean instances from the pool.

1.2.4 JBoss JPA Test

The Java Persistence API (JPA) is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database. JPA itself is just a specification, not a product; it cannot perform persistence or anything else by itself. JPA is just a set of interfaces, and requires an implementation. JPA also requires a database to persist to. The JPA allows POJO (Plain Old Java Objects) to be easily persisted without requiring the classes to implement any interfaces or methods. JPA allows an object's object-relational mappings to be defined through standard annotations or XML defining how the Java class maps to a relational database table. JPA also defines a runtime EntityManager API for processing queries and transaction on the objects against the database. JPA defines an object-level query language, JPQL, to allow querying of the objects from the database.

An entity is a lightweight persistence domain object. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes.

The persistent state of an entity is represented through either persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

The JPA persistence context contains the entities managed by the persistence provider. The persistence context acts like a first level (transactional) cache for interacting with the datasource. Loaded entities are placed into the persistence context before being returned to the application. Entities changes are also placed into the persistence context (to be saved in the database when the transaction commits).

Collection-valued persistent fields and properties must use the supported Java collection interfaces regardless of whether the entity uses persistent fields or properties. The following collection interfaces may be used:

- java.util.Collection
- java.util.Set
- java.util.List
- java.util.Map

If the entity class uses persistent fields, the type in the preceding method signatures must be one of these collection types. Generic variants of these collection types may also be used. While an administrator is monitoring an application in real-time, he/she may not be aware of a sudden

slowdown/malfunctioning of the application due to technical glitches at the backend. This may be due to a sudden change in the entities and collections of the Java Persistence API which when left unnoticed will render the application inaccessible to users. To avoid such a situation, you can use the **JBoss JPA** test. For each entity of an application that is to be monitored, this test reports the number of times each entity was loaded, inserted, fetched, updated, etc. In due course, this test also reports the number of times the collection of each entity was loaded, inserted, deleted, fetched,etc. This way administrators can proactively be alerted to technical failures of the applications and rectify the same before end users start complaining!

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each *Application:entity* of the target WildFly JBoss that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer

to Section **1.1** of this document.

9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Entity Load count:	Indicates the number of times this entity was loaded.	Number	Comparing the value of this measure across the entities will help you identify the entity that is the busiest - in terms of loading on the application.
Entity Insert count:	Indicates the number of times this entity was inserted.	Number	
Entity Fetch count:	Indicates the number of times this entity was fetched.	Number	
Entity Update count:	Indicates the number of times this entity was updated.	Number	
Entity Delete count:	Indicates the number of times this entity was deleted.	Number	
Entity Failure count:	Indicates the number of times this entity failed.	Number	A low value is desired for this measure. A sudden/gradual increase in the value of this measure is a cause of concern as this may lead to poor user experience.
Collection Load count:	Indicates the number of times the collection was loaded on this entity.	Number	
Collection Recreate count:	Indicates the number of times the collection was	Number	

Measurement	Description	Measurement Unit	Interpretation
	recreated for this entity.		
Collection Fetch count:	Indicates the number of times the collection was fetched for this entity.	Number	
Collection Update count:	Indicates the number of times the collection was updated for this entity.	Number	
Collection Remove count:	Indicates the number of times the collection was removed from this entity.	Number	

1.2.5 JBoss MQ Queues Test

Clients that are in the point-to-point paradigm typically use queues. They expect that message sent to a queue will be received by only one other client once and only once. If multiple clients are receiving messages from a single queue, the messages will be load balanced across the receivers. Queue objects, by default, will be stored under the JNDI queue/ sub context.

This test auto-discovers the queues on the WildFly JBossserver, and monitors each queue for the size, number, and type of messages it holds, so that impending overloads and probable delivery bottlenecks can be proactively isolated and corrected.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each queue of the target WildFly JBoss server that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured

3. **PORT**– The port at which the specified **HOST** listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a *master* or a *slave* in a JBoss cluster. By default, *none* will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, *none* will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Max queue depth:	Indicates the max value of the number of messages that were in this queue since the start of the queue.	Number	
Current queue depth:	Indicates the number of messages that are currently in this queue.	Number	A high value is indicative of server workload, or a delivery bottleneck.
Queue occupied:	Indicates the percentage of queue length that is	Percent	A high value is a cause for concern as it could indicate a bottleneck in

Measurement	Description	Measurement Unit	Interpretation
	occupied by messages.		message delivery, which may be heavily populating the queue with messages.
Messages delivered:	Indicates the number of messages in this queue that have been delivered.	Number	A high value is desired for this measure.
Messages scheduled:	Indicates the number of messages in this queue that are currently scheduled to be delivered.	Number	
Messages added:	Indicates the number of messages that were added to this queue during the last measurement period.	Number	
Last message sent time:	Indicates the time that elapsed since the last message was sent from this queue.	Secs	Ideally, the value of this measure should be low. A high value is indicative of a delay in message delivery or a message processing bottleneck.
Subscriber count:	Indicates the number of subscribers registered with this queue.	Number	
Receivers count:	Indicates the number of clients who are currently configured as receivers of messages from this queue.	Number	
Message processing rate:	Indicates the rate at which messages are being processed by this queue.	Msgs/sec	

1.2.6 JBoss MQ Topics Test

Topics are used in the publish-subscribe paradigm. When a client publishes a message to a topic, he/she expects that a copy of the message will be delivered to each client that has subscribed to the topic. However, if the client is not up, running and receiving messages from the topics, it will miss messages published to the topic. To get around this problem of missing messages, clients can start a durable subscription. This is like having a VCR record a show you cannot watch at its scheduled time so that you can see what you missed when you turn your TV back on. Similarly, m*essages meant for a durable subscriber are stored in the persistent cache even when the subscriber is inactive. These messages are delivered to durable subscribers when they connect to the server. Using the* **JBoss MQ Topics** test, continuous monitoring of the topics is possible. This test auto discovers the topics on the WildFly JBoss server and reports the number of durable messages and the durable subscribers of the topic. This test also helps the administrators to keep track on the messages added into each topic and the messages delivered from each topic along with the processing rate of the messages from the topic. This way, administrators can figure out the topic that is currently subscribed by most of the subscribers!

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each topic of the WildFly JBoss server that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave

in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.

7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Durable messages count:	Indicates the number of durable messages currently in the topic.	Number	
Durable subject count:	Indicates the number of durable subscribers to the topic.	Number	Messages meant for a durable subscriber are stored in the persistent cache even when the subscriber is inactive. These messages are delivered to durable subscribers when they connect to the server.
Max topic depth:	Indicates the maximum value of the number of messages in the topic since the start of the queue.	Number	
Message processing rate:	Indicates the rate at which messages were being processed by the topic in the last measurement period.	Msgs/Sec	

Measurement	Description	Measurement Unit	Interpretation
Non- durable messages count:	Indicates the number of non- durable messages currently in the topic.	Number	If the total number of durable subscribers is high, then we can expect the total durable messages to be stored on the server also to be relatively on the higher side.
Non- durable subject count:	Indicates the number of subscribers to the topic who are currently non-durable.	Number	
Total subject count:	Indicates the total number of subscribers to a topic.	Number	
Messages added count:	Indicates the number of messages that were added to this topic during the last measurement period.	Number	
Message delivered count:	Indicates the number of messages in this topic that have been delivered.	Number	A high value is desired for this measure.

1.2.7 JBoss Servlet Test

This test monitors the servlets deployed on the WildFly JBoss server and reports the following:

- The number of incoming requests to each servlet;
- The time taken to load each servlet;
- The time taken by each servlet to to process the requests and the rate at which the requests are processed

This way, the administrators can identify the servlet that is handling the maximum number of requests.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each servlet of the WildFly JBoss server that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e., if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.
6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Request count:	Indicates the number of incoming requests that were handled by this	Number	This value is a good indicator on the load of the servlet.

Measurement	Description	Measurement Unit	Interpretation
	servlet.		
Load time:	Indicates the total time taken to load this servlet.	Secs	A low value is desired for this measure.
Processing time:	Indicates the time taken to process the incoming requests to this servlet during the last measurement period.	Secs	If the value taken to process the requests is higher, then it may be due to a request taking too long to be processed, a connection issue where the servlet failed to load, etc.
Request processing rate:	Indicates the rate at which the requests were processed by this servlet during the last measurement period.	Number/Sec	A high value is desired for this measure.

1.2.8 JBoss Transactions Test

A transaction is a unit of work containing one or more operations involving one or more shared resources having ACID properties. ACID is an acronym for atomicity, consistency, isolation and durability, the four important properties of transactions. These terms are defined below in detail:

- **Atomicity**: A transaction must be atomic. This means that either all the work done in the transaction must be performed, or none of it must be performed. Doing part of a transaction is not allowed.
- **Consistency**: When a transaction is completed, the system must be in a stable and consistent condition.
- **Isolation**: Different transactions must be isolated from each other. This means that the partial work done in one transaction is not visible to other transactions until the transaction is committed, and that each process in a multi-user system can be programmed as if it was the only process accessing the system.
- **Durability**: The changes made during a transaction are made persistent when it is committed. When a transaction is committed, its changes will not be lost, even if the server crashes afterwards.

Transactions are performed using the Java transaction API. The Java Transaction API consists of three elements: a high-level application transaction demarcation interface, a high-level transaction manager interface intended for application server, and a standard Java mapping of the X/Open XA protocol intended for transactional resource manager. All of the JTA classes and interfaces occur within the **javax.transaction** package, and the corresponding JBossJTA implementations within the **com.arjuna.ats.jta** package. The **UserTransaction** interface provides applications with the ability to control transaction boundaries. It has methods for beginning, committing, and rolling back top-level transactions. The **TransactionManager** interface allows the application server to control transaction boundaries on behalf of the application being managed.

The Transaction Manager maintains the transaction context association with threads as part of its internal data structure. A thread's transaction context is either null or it refers to a specific global transaction. Multiple threads may be associated with the same global transaction.

Each transaction context is encapsulated by a Transaction object, which can be used to perform operations which are specific to the target transaction, regardless of the calling thread's transaction context. Sometimes, there may be too many transactions being performed on the target application server and administrators may lose count on the number of transactions of each type. This is exactly where the **JBoss Transactions** test helps!

This test measures the transaction activity performed by the WildFly JBoss server. Using this test, administrators can easily track the numerical statistics of each transaction type and identify which type of transaction is executed too often on the target WildFly JBoss server.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for the target WildFly JBoss server that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is

currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.

6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Number of transactions:	Indicates the number of transactions on this server.	Number	This measure is a good indicator of the load on the server.
Aborted transactions:	Indicates the total number of transactions that were aborted i.e., interrupted in this server during the last measurement period.	Number	
Application transactions rolled back:	Indicates the number of transactions that were rolled back due to application errors during the last measurement period.	Number	
Committed	Indicates the number of	Number	If the number of transactions that

Measurement	Description	Measurement Unit	Interpretation
transactions:	transactions that were committed to be processed by the Transaction manager during the last measurement period.		are being committed is very high, it signifies load on the server. It might be caused when some locked transactions are released suddenly.
Heuristics transactions:	Indicates the number of transactions that were terminated with heuristic outcome during the last measurement period.	Number	A heuristic completion (or heuristic decision) occurs when a resource makes a unilateral decision during the completion stage of a distributed transaction to commit or rollback updates. This can leave distributed data in an indeterminate state. Network failures or resource timeouts are possible causes for heuristic completion. In the event of an heuristic completion, one of the following heuristic outcome exceptions may be thrown: **HeuristicRollback**—one resource participating in a transaction decided to autonomously rollback its work, even though it agreed to prepare itself and wait for a commit decision. If the Transaction Manager decided to commit the transaction, the resource's heuristic rollback decision was incorrect, and might lead to an inconsistent outcome since other branches of the transaction were committed. **HeuristicCommit**—one resource participating in a transaction decided to autonomously commit its

Measurement	Description	Measurement Unit	Interpretation
			work, even though it agreed to prepare itself and wait for a commit decision. If the Transaction Manager decided to rollback the transaction, the resource's heuristic commit decision was incorrect, and might lead to an inconsistent outcome since other branches of the transaction were rolled back. **HeuristicMixed**—the Transaction Manager is aware that a transaction resulted in a mixed outcome, where some participating resources committed and some rolled back. The underlying cause was most likely heuristic rollback or heuristic commit decisions made by one or more of the participating resources. **HeuristicHazard**—the Transaction Manager is aware that a transaction might have resulted in a mixed outcome, where some participating resources committed and some rolled back. But system or resource failures make it impossible to know for sure whether a Heuristic Mixed outcome definitely occurred. The underlying cause was most likely heuristic rollback or heuristic commit decisions made by one or more of the participating resources.
Inflight	Indicates the number of	Number	A significantly high value may

Measurement	Description	Measurement Unit	Interpretation
transactions:	transactions that have begun but are yet to be terminated during the last measurement period.		denote a load on the server. This may indicate that specific transactions are taking too long to process requests.
Nested transactions:	Indicates the number of nested i.e., sub transactions that were created during the last measurement period.	Number	
Resource transactions rolled back:	Indicates the number of transactions that rolled back due to resource failure during the last measurement period.	Number	A high value indicates a problem with the application or with some other dependent servers (e.g. Database).
Timed- out transactions:	Indicates the number of transactions that rolled back due to timeout during the last measurement period.	Number	A steady increase in the value of this measure could be due to the problem with the application or with some other dependent server like the database.

1.2.9 JBoss XA-Datasources Test

An XA Datasource is needed to execute a distributed transaction. Generally, a distributed transaction spans 2 or more different datasources. An XA datasource is used instead of the datasource if the target Jboss application:

- Uses the Java Transaction API (JTA)
- Includes multiple database updates within a single transaction
- Accesses multiple resources, such as a database and the Java Messaging Service (JMS), during a transaction
- Uses the same connection pool on multiple servers

Though the XA datasource can use all transaction features of the application server and need fewer physical connections, the XA datasource involves performance overhead along with the risk of a higher deadlock. There may also be a necessity of more physical connections when an emulated XA datasource is used in the target environment. When an XA datasource is used in the target environment, administrators would be required to carefully monitor the XA datasource as the slightest variation in the connections or a higher waiting time for a connection may cause a severe performance overhead on the XA datasource. To overcome such abnormalities, administrators may use the **JBoss XA-Datasources** test. This test helps the administrators figure out the following:

- The number of connections that are currently available and the number of active connections;
- The number of connections that were created and destroyed using this datasource;
- The time taken to create a connection on this datasource and the maximum time taken for creating a connection;
- How many connections are available in a connection pool and how well the connections of a connection pool are utilized?
- The prepared statement cache size and the number of prepared statements added
- How many times prepared statements are accessed and how well the prepared statement cache caters to the requests for accessing the prepared statements?

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each XA datasource of the target WildFly JBoss that is to be monitored

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST** – The host for which the test is to be configured
3. **PORT**– The port at which the specified **HOST**listens. By default, this is 9990.
4. **SSL** – If the WildFly JBoss server being monitored is an SSL-enabled server, then set the **SSL**flag to **Yes**. If not, then set the **SSL**flag to **No**.
5. **IS JBOSS RUNNING IN DOMAIN MODE?**– Specify whether the server to be monitored is currently running in **DOMAIN MODE** or not. By default, this flag is set to **No** which implies that the server is currently running in **STANDALONE MODE**. If you have started the target JBoss server using the default web profile configuration in domain mode i.e, if you have executed the **./domain.sh** command

from the **<JBoss_INSTALL_DIR>/bin** directory, then specify **Yes** against this flag.

6. **JBOSS HOST NAME** – Specify whether the target server to be monitored is a master or a slave in a JBoss cluster. By default, none will be specified here which implies that the target JBoss server is a standalone server. Refer to Section **1.1.1** to know how to identify whether the target server is a master or slave in your environment.
7. **JBOSS SERVER INSTANCE NAME**– Specify the name of the server instance that is to be monitored. By default, none will be specified here. Refer to Section **1.1.1** to identify the name of the server instance that is to be monitored.
8. **MANAGEMENT USER** and **MANAGEMENT PASSWORD**– Specify the credentials of the user who is authorized to access the management console of the target JBoss server. To create a new user, refer to Section **1.1** of this document.
9. **CONFIRM PASSWORD**– Confirm the **MANAGEMENT PASSWORD** by retyping it here.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
Active count:	Indicates the number of active connections in this XA datasource.	Number	A high value is desired for this measure.
Available count:	Indicates the number of connections that are currently available for use in this XA datasource.	Number	A low value for this measure indicates that adequate connections are not available in the datasource.
Average blocking time:	Indicates the average time spent for blocking a connection during the last measurement period.	Secs	
Average creation	Indicates the	Secs	Comparing the value of this measure

Measurement	Description	Measurement Unit	Interpretation
time:	average time spent for creating a physical connection during the last measurement period.		across the XA datasources will reveal the XA datasource that is the slowest when creating a physical connection.
Created count:	Indicates the number of connections that were created in this XA datasource.	Number	
Destroyed count:	Indicates the number of connections that were destroyed in this XA datasource since the start of the server.	Number	
Max creation time:	Indicates the maximum time taken to create a connection in this XA datasource.	Secs	Compare the value across the XA datasources to identify the XA datasource that is taking too long to create a connection i.e., you can identify the XA datasource that is the slowest.
Max used count:	Indicates the maximum number of connections that were used in this XA datasource.	Number	
Max wait time:	Indicates the maximum time a user has to wait for a connection in this XA datasource.	Secs	A low value is desired for this measure. An abnormally high value or a sudden increase in the value of this measure is a cause of concern which requires further investigation.

Measurement	Description	Measurement Unit	Interpretation
Timed out:	Indicates the time taken for a connection to time out.	Mins	Normally, a connection would time out if it has been waiting in the queue for a longer time.
Total blocking time:	Indicates the total time taken for a connection to fail or time out.	Secs	
Total creation time:	Indicates the time taken to create a connection in this XA datasource.	Secs	A low value is desired for this measure.
Maximum pool size:	Indicates the maximum number of connections allowed in the connection pool of this XA datasource.	Number	The `max-pool-size` parameter defines the maximum size of the connection pool. This parameter is more important because it limits the number of active connections to the datasource and thus the concurrent activity on the data source. If this value is set too low it's likely that the platform's hardware resources will be underutilized.
Minimum pool size:	Indicates the minimum number of connections that the connection pool of this XA datasource can contain.	Number	The `min-pool-size` data source parameter defines the minimum size of the connection pool. The default minimum is zero connections, so if a minimum pool size is not specified, no connections will be created in the connection pool when the platform starts. As data source transactions occur, connections will be requested but because the pool defaults to zero on start up, none will be available. The connection pool examines the

Measurement	Description	Measurement Unit	Interpretation
			minimum and maximum parameters, creates connections and places them in the pool. Users of any affected application will experience a delay while this occurs. During periods of inactivity the connection pool will shrink, possibly to the minimum value, and later when transactions occur application performance will again suffer.
PreparedStatement cache size:	Indicates the number of prepared statements per connection to be kept open and reused in subsequent requests.	Number	When using the JPA annotations for queries, the result is prepared statements that will be executed against the database. Prepared statements have two phases for execution: preparation of the statement, and execution of the statement. Statement preparation involves significant CPU load so to improve throughput prepared statements can be cached. Statements can be cached either via the JDBC driver or configuration of the data source. To enable caching of prepared statement, add the following two lines to the data source configuration file, a file with the pattern *-ds.xml (where the * is usually your database, such as oracle, mysql, db2, etc.) in the directory: `JBOSS_ EAP_ DIST/jboss-as/server/PROFILE/deploy` . Note that the minimal configuration does not support data sources.

Measurement	Description	Measurement Unit	Interpretation

			`<prepared-statement-cache-size>100</prepared-statement-cache-size>`
			`<shared-prepared-statements>true</shared-prepared-statements>`
			The first line enables the prepared statement cache and sets the size of the cache. This should be large enough to hold all the prepared statements across any and all deployed applications using this particular data source (multiple data sources can be configured). The second line states that if two requests are executed in the same transaction the prepared statement cache should return the same statement. This will avoid the use of CPU cycles in preparing the same statements over and over again, increasing throughput and decreasing response times. The actual improvement will vary according to specific circumstances but is well worth the effort.

PreparedStatement cache access count:	Indicates the number of times the prepared statement cache was accessed.	Number	
PreparedStatement cache add count:	Indicates the number of new prepared statements that were added i.e.,	Number	

Measurement	Description	Measurement Unit	Interpretation
	cached in the prepared statement cache.		
PreparedStatement cache current size:	Indicates the number of prepared statements that are currently available in the cache.	Number	If the value of this measure is close to the *PreparedStatement cache size* measure, then it indicates that the prepared statement cache is almost running out of space.
vPreparedStatement cache delete count:	Indicates the number of prepared statements that are deleted from the prepared statement cache.	Number	
PreparedStatement cache hit count:	Indicates the number of times the statement was accessed from the prepared statement cache.	Number	A high value is desired for this measure.
PreparedStatement cache miss count:	Indicates the number of times a request for a statement was not serviced from the prepared statement cache.	Number	A high value for this measure is a cause of concern.

1.2.10 Web Service Test

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like

the Internet in a manner similar to inter-process communication on a single computer. A complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of the following:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

The following are the major uses of the Web Services:

- **Reusable application-components**: Often applications need repeated access to application-components like currency conversion, weather reports, or even language translation. In such cases, the web services can be used to offer the application-components as services with ease.
- **Connect existing software**: Web services can help to solve the interoperability problem by giving different applications a way to link their data. With Web services you can exchange data between different applications and different platforms. Any application can have a Web Service component. Web Services can be created regardless of programming language.

In certain environments, administrators are required to keep an eye on the web services that offer repeated access to the application-components i.e., operations so that the work load on the users using those application components can be minimized. If for some reason the web service takes too long to respond or is unavailable to cater to the needs of the users, then the users will be deprived of access to the application-components involved in that particular web service. To avoid such inconvenience caused to the users, administrators are required to continuously monitor the web

services. The **Web Service** test helps administrators to perform this task perfectly. By continuously monitoring each operation i.e., application component of a web service that is offered, using the SOAP commands, this test helps administrators identify the availability, response time and response code of the web service and quickly figure out discrepancies if any web service is deemed unavailable. This way, the web services can be kept available round the clock thus helping the users perform their tasks without any difficulty.

Target of the test : A WildFly JBoss server

Agent deploying the test : An internal agent

Outputs of the test : One set of results for each *WebService:Operation* i.e., application-component performed on the target server that is being monitored.

Configurable parameters for the test

1. **TEST PERIOD** - How often should the test be executed
2. **HOST -** The host for which the test is to be configured
3. **PORT -** The port number at which the specified **HOST** listens
4. **WSDL URL** - This test emulates a user accessing a specific web service(s) on the target server to determine the availability and responsiveness of the server. to enable this emulation, you need to configure the test with the url of the web service that it should access. specify this url against the **WSDL URL** parameter. if required, you can even configure multiple WSDL URLs - one each for every web service that the test should attempt to access. if each WSDL URL configured requires special permissions for logging in, then, you need to configure the test with separate credentials for logging into every WSDL URL. likewise, you need to provide instructions to the test on how to validate the content returned by every WSDL URL, and also set an encoding format for each wsdl url. to enable administrators to easily configure the above per WSDL URL, eg enterprise provides a special interface. to access this interface, click on the encircled '+' button alongside the url text box in the test configuration page. alternatively, you can even click on the encircled '+' button adjacent to the WSDL URL parameter in the test configuration page. to know how to use this special interface, refer to Section **1.2.10.1**.
5. **OPERATIONS** – Once the WSDL URL(s) are specified, the operations that are offered by the web services and those that are to be monitored have to be configured. To select the required operations for monitoring, eG Enterprise provides a special interface. TO access this interface, click on the encircled '+' button alongside the **OPERATIONS** text box in the test configuration page. Alternatively, you can even click on the encircled '+' button adjacent to the **OPERATIONS**

parameter in the test configuration page. To know how to use this special interface, refer to Section **1.2.10.1**.

6. **TIMEOUT** – Specify the duration (in seconds) for which this test should wait for a response from the server. If there is no response from the server beyond the configured duration, the test will timeout. By default, this is set to 30 seconds.
7. **DETAILED DIAGNOSIS** - To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option.

   The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:

   - The eG manager license should allow the detailed diagnosis capability
   - Both the normal and abnormalfrequencies configured for the detailed diagnosis measures should not be 0.

Measures made by the test:

Measurement	Description	Measurement Unit	Interpretation
WSDL url availability:	Indicates whether the web service was able to respond successfully to the query made by the test.	Percent	Availability failures could be caused by several factors such as the web service process (es) being down, the web service being misconfigured, a network failure, etc. Temporary unavailability may also occur if the web service is overloaded. Availability is determined based on the response code returned by the service. A response code between 200 to 300 indicates that the service is available.
WSDL response time:	Indicates the time taken by the eG agent to get	Secs	Response time being high denotes a problem. Poor response times

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td>the configured web service.</td><td></td><td>may be due to the service being overloaded or misconfigured. If the URL accessed involves the generation of dynamic content by the service, backend problems (e.g., an overload at the application server or a database failure) can also result in an increase in response time.</td></tr>
<tr><td>Port status:</td><td>Indicates whether/not the port of the web server is reachable.</td><td></td><td>The values reported by this measure and the corresponding numeric equivalents are listed in the table below:
<table><tr><th>Measure Values</th><th>Numeric Values</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr></table>
Note:
By default, this measure reports the above-mentioned Measure Values to indicate whether the server has been rebooted or not. In the graph of this measure however, the Measure Values are represented using the numeric equivalents only.</td></tr>
<tr><td>TCP connection availability:</td><td>Indicates whether the test managed to establish a TCP connection to the server.</td><td>Percent</td><td>Failure to establish a TCP connection may imply that either the web server process is not up, or that the process is not operating correctly. In some cases of extreme</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			overload, the failure to establish a TCP connection may be a transient condition. As the load subsides, the server may start functioning properly again.
TCP connect time:	This measure quantifies the time for establishing a TCP connection to the web server host.	Secs	Typically, the TCP connection establishment must be very small (of the order of a few milliseconds). Since TCP connection establishment is handled at the OS-level, rather than by the application, an increase in this value signifies a system-level bottleneck on the host that supports the web server.
Server response time:	Indicates the time period between when the connection was established and when the web server sent back a response header to the client.	Secs	While the total response time may depend on several factors, this measure is typically, a very good indicator of a server bottleneck (e.g., because all the available server threads or processes are in use).
Response code:	The response code returned by the web server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (eg., page not found). A 5xx value indicates a server error.
Service availability:	Indicates whether/not the web service is available.	Percent	A value of 100 indicates that the web service is available and a value of 0 indicates that the web service is not available.
Operation status:	Indicates whether/not the configured operation is present in		This measure will not report metrics if the **OPERATION** parameter in the test configuration page is none in

Measurement	Description	Measurement Unit	Interpretation
	the web service.		the test configuration page.
Operation Content length:	Indicates the response code returned by the server for the simulated request.	Number	A value between 200 and 300 indicates a good response. A 4xx value indicates a problem with the requested content (e.g., page not found). A 5xx value indicates a server error. This measure will not report metrics if the **OPERATION** parameter in the test configuration page is none or if an invalid **Value** is specified or if the Value is not specified in the **HTML View** tab while configuring the operation for monitoring in the test configuration page.
Operation Content validity:	This measure validates whether the operation was successful in executing the request made to it.	Percent	A value of 100% indicates that the content returned by the test is valid. A value of 0% indicates that the content may not be valid. This capability for content validation is especially important for multi- tier web applications. For example, a user may not be able to login to the web site but the server may reply back with a valid HTML page where in the error message, say, “Invalid Login” is reported. In this case, the availability will be 100 % (since we got a valid HTML response). If the test is configured such that the content parameter should exclude the string “Invalid Login”, in the above scenario content validity would have a value 0.

Measurement	Description	Measurement Unit	Interpretation
			This measure will not report metrics if the **OPERATION** parameter in the test configuration page is none or if an invalid **Value** is specified or if the Value is not specified in the **HTML View** tab while configuring the operation for monitoring in the test configuration page.
Operation execution time:	Indicates the time taken to invoke the configured operation in the web service.	Secs	This measure will not report metrics if the **OPERATION** parameter in the test configuration page is none or if an invalid **Value** is specified or if the Value is not specified in the **HTML View** tab while configuring the operation for monitoring in the test configuration page.

1.2.10.1 Configuring Multiple WSDL URLs for Monitoring

In order to enable the eG agent to connect to multiple WSDL URLs and pull out the required metrics from them, the eG administrative interface provides a special page using which different WSDL URLs and their corresponding operations that need to be monitored can be specified. To configure the WSDL URLs, do the following:

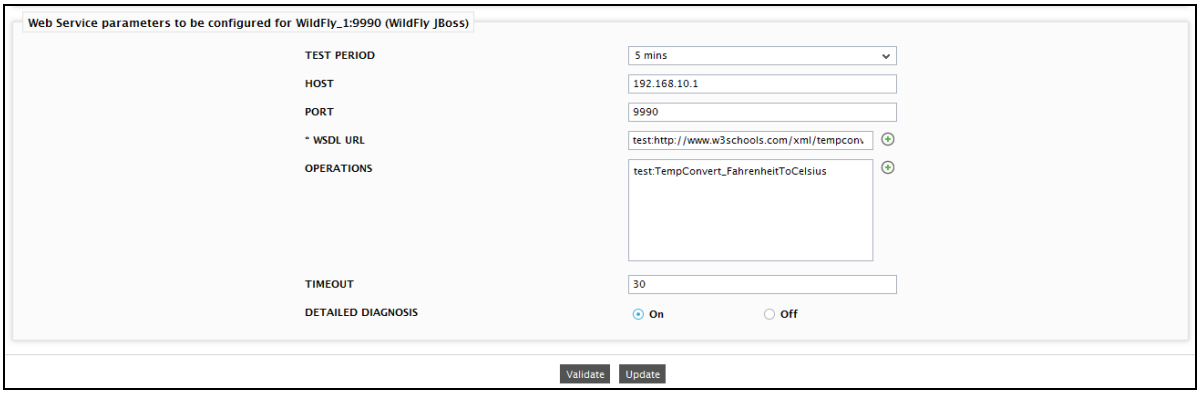

Figure 1.5: Configuring the WebService test

1. Click on the encircled '+' button alongside the **WSDL URL** text box. 1.2.10 will then appear.

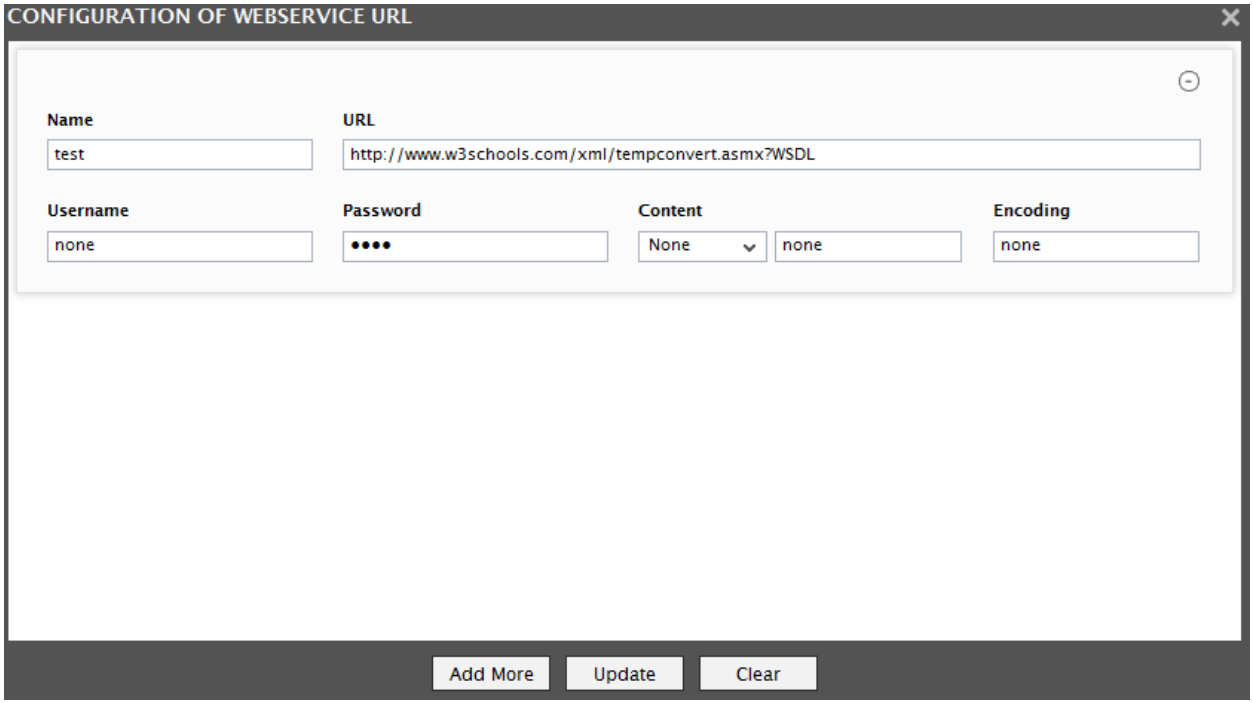

Figure 1.6: The WebService URL Configuration page

2. Specify the following in Figure 1.6:

   - **Name:** Specify a unique name by which the WSDL URL you will be specifiying shortly will be referred to across the eG user interface. This is the name that will appear as the descriptor of this test.
   - **URL:** Enter the WSDL URL of the web service that this test should access.
   - **Username** and **Password:** These parameters are to be set only if a specific user name / password has to be specified to login to the web service (i.e., WSDL URL) that you have configured for monitoring. In this case, provide valid login credentials using the **Username** and **Password** text boxes. If the server on which **WebService** test executes supports 'Anonymous user access', then these parameters will take either of the following values:
     - A valid **Username** and **Password** for the configured **WSDL URL**
     - none in both the **Username** and **Password** text boxes of the configured WSDL URL, if no user authorization is required
     - Some servers however, support NTLM (Integrated Windows) authentication, where valid login credentials are mandatory. In other words, a none specification will not be supported

by such servers. Therefore, in this case, against each configured **WSDL URL**, you will have to provide a valid **Username** in the format: domainname\username, followed by a valid **Password**.

- Please be sure to check if your web service requires HTTP authentication while configuring this parameter. HTTP authentication typically involves a separate pop-up window when you try to access the page. Many services use HTTP POST for obtaining the user name and password and validating the user login. In such cases, the username and password have to be provided as part of the POST information and NOT as part of the **CREDENTIALS** specification for the WebService test.

- **Content** : The **Content** parameter has to be configured with an instruction:value pair that will be used to validate the content being returned by the test. If the **Content** value is None, no validation is performed. On the other hand, if you pick the Include option from the **Content** list, it indicates to the test that for the content returned by the web server to be valid, the content must include the specified value (a simple string search is done in this case). This value should be specified in the adjacent text box. Similarly, if the Exclude option is chosen from the **Content** drop-down, it indicates to the test that the server's output is valid if it does not contain the value specified in the adjacent text box. The Include or Exclude value you specify in the text box can include wildcard characters. For example, an Include instruction can be *Home page*.

- **Encoding**: Sometimes the eG agent has to parse the **WSDL URL** content with specific encoding other than the default (ISO-8859-1) encoding. In such a case, specify the type of encoding using which the eG agent can parse the **WSDL URL** content in the **Encoding** text box. By default, this value is none.

3. Similarly, you can add multiple URL specifications by clicking the **Add More** button. To remove a WSDL URL specification, click on the encircled '-' button corresponding to it. To clear all **WSDL URL** specifications, click the **Clear** button. To update all the changes you made, click the **Update** button.
4. Once **Update** is clicked, you will return to the test configuration page as shown in Figure 1.5. The **WSDL URL** text box in the test configuration page will display just the **Name**s - i.e., the unique display names - that you may have configured for the multiple WSDL URLs, as a comma-separated list. To view the complete WSDL URL specification, click the encircled '+' button alongside the **WSDL URL** text box, once again.

1.2.10.2 Configuring Multiple Operations for Monitoring - WebServiceTest

By default, the **WebServiceTest** will be configured with the WSDL URLs that offer the web services that are to be monitored. To configure the operations that are offered by the WSDL URLs, do the

following:

1. Click on the encircled '+' button alongside the **OPERATIONS** text box as shown in Figure 1.5. Figure 1.7 will then appear.

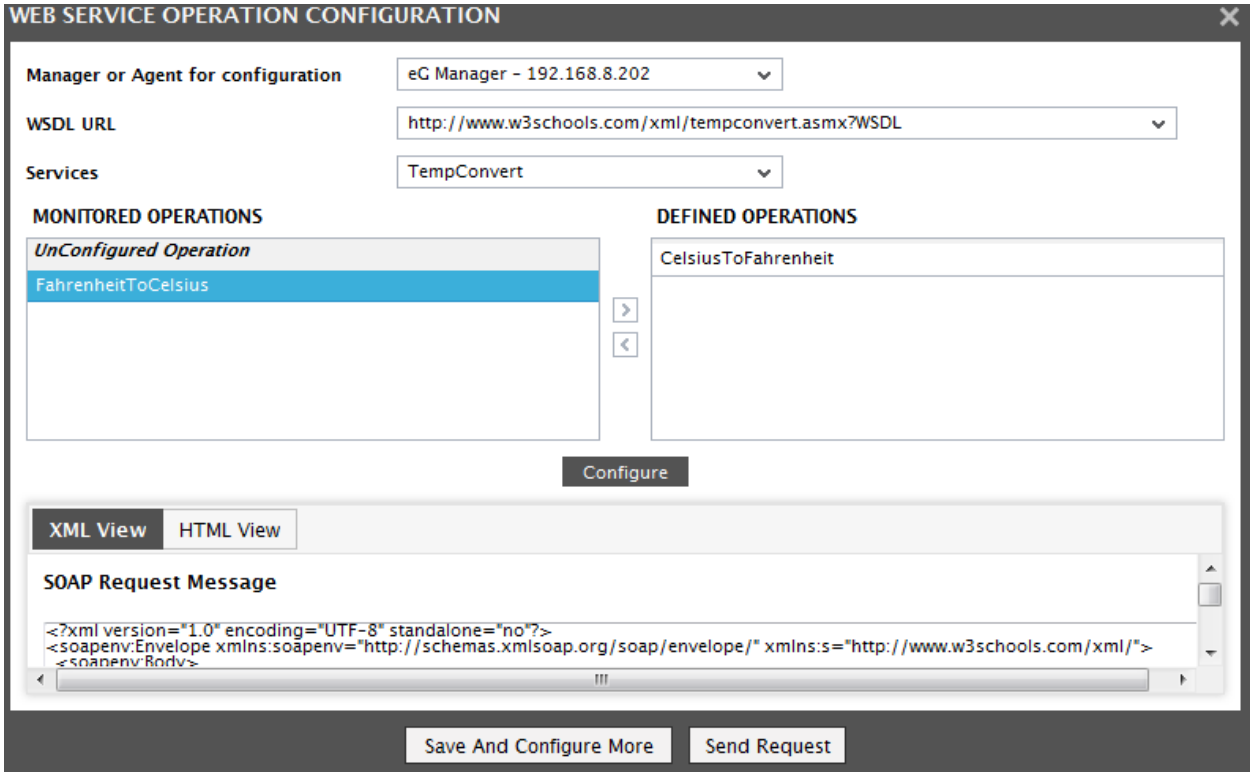

Figure 1.7: Configuring the Web Service Operation

2. Specify the following in Figure 1.7:

   - **Manager/Agent for accessing WSDL URL:** Select the eG agent or the eG Manager that is authorized to access the configured WSDL URL from this list.
   - **WSDL URL:** Once the eG agent/eG Manager is chosen from the **Manager/Agent for accessing WSDL URL** list, this list will be populated automatically with all the WSDL URLs specified in the **WSDL URL** text box (See Figure 1.5). Select the **WSDL URL** of your choice from this list.
   - **Services:** The web services offered by the chosen WSDL URL will then be populated in this list. Select a service of your choice from this list.
     - The operations that are offered by the chosen service will then be populated in the **DEFINED OPERATIONS** list. To monitor a chosen operation, select the operation and click the **<** button. This will move the chosen operation to the **MONITORED OPERATIONS** list.

- Click the **Configure** button to save the changes.
- The eG agent uses SOAP requests to obtain the necessary metrics from the web service. Once the operation is configured, the XML View of the SOAP Request corresponding to the chosen operation will be generated and listed in the **XML View** tab. Likewise, the **HTML View** tab lists the **SOAP Parameter** that is passed to collect the required metrics for the chosen operation.
- To obtain operation-level statistics, it is important to specify a valid value in the **VALUE** text box of the **HTML VIEW** tab as shown in Figure 1.8. Each time the test is executed, this value will be provided as an input to the chosen operation.

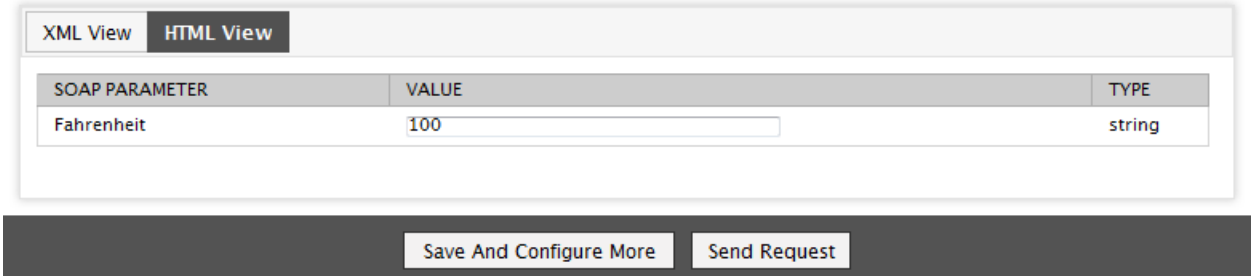

Figure 1.8: Specifying the value for the chosen operation

- Click the **Save and Cofigure More** button to save the changes made.
- If you wish to verify if the **VALUE** specified in the **HTML View** tab is valid, then you can do so by clicking the **Send Request** button. Figure 1.9 will then appear. If the value specified in the **VALUE** text box is indeed valid, then the operation will be performed on the value and the result will be specified. For example, if your chosen operation is FahrenheittoCelsius, the **SOAP PARAMETER** is Farenheit and the value that you wish to convert is 100, the result will be specified in the **WEB SERVICE RESPONSE** pop up window as below: *<FahrenheitToCelsiusResult>37.7777777777778</FahrenheitToCelsiusResult>*

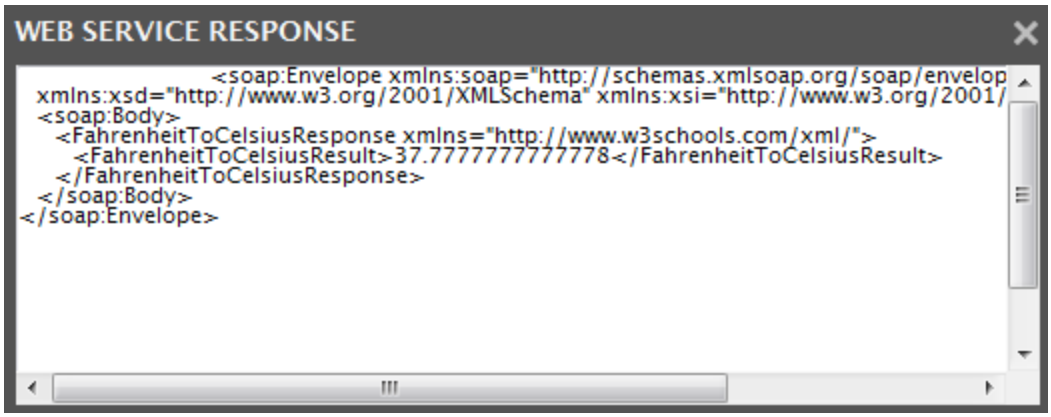

Figure 1.9: The value that appears when the operation is performed successfully

If you have specified an invalid value, then a message as follows will be displayed in the pop up window:

<FahrenheitToCelsiusResult>Error</FahrenheitToCelsiusResult>

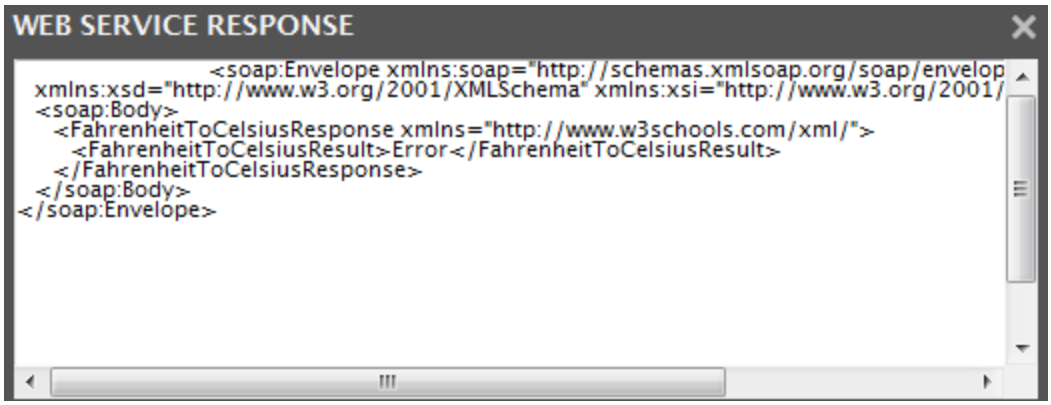

Figure 1.10: An Error appearing during value conversion

If you do not specify a **VALUE** or specify an invalid value, operation-level statistics will not be collected by the eG agent and such metrics will not be available in the eG monitoring interface.

3. Similarly, you can configure multiple Operations by clicking the **Configure** button in Figure 1.7. To remove an operation, select the operation from the **MONITORED OPERATION** list and click the **>** button.
4. Once **Save and Configure More** button is clicked, you will return to the test configuration page (see Figure 1.5). The **OPERATIONS** text box in the test configuration page will display just the operations, as a comma-separated list. To view the complete operation specification, click the encircled '+' button alongside the **OPERATIONS** text box, once again.

1.3 The Java Transactions Layer

By default, this layer will not be available for the WildFly JBoss server. This is because, the **Java Business Transactions** test mapped to this layer is disabled by default. To enable the test, follow the *Agents -> Tests -> Enable/Disable* menu sequence, select *WildFly JBoss* as the **Component type**, *Performance* as the **Test type**, and then select **Java Business Transactions** from the **DISABLED TESTS** list. Click the **Enable** button to enable the selected test, and click the **Update** button to save the changes.

1.3.1 Java Business Transactions Test

The responsiveness of a transaction is the key determinant of user experience with that transaction; if response time increases, user experience deteriorates. To make users happy, a Java business transaction should be rapidly processed by each of the JVM nodes in its path. Processing bottlenecks on a single JVM node can slowdown/stall an entire business transaction or can cause serious transaction errors. This in turn can badly scar the experience of users. To avoid this, administrators should promptly identify slow/stalled/errored transactions, isolate the JVM node on which the slowness/error occurred, and uncover what caused the aberration on that node – is it owing to SQL queries executed by the node? Or is it because of external calls – eg., async calls, SAP JCO calls, HTTP calls, etc. - made by that node? The **Java Business Transactions** test helps with this!

This test runs on a BTM-enabled JVM in an IT infrastructure, tracks all the transaction requests received by that JVM, and groups requests based on user-configured pattern specifications. For each transaction pattern, the test then computes and reports the average time taken by that JVM node to respond to the transaction requests of that pattern. In the process, the test identifies the slow/stalled transactions of that pattern, and reports the count of such transactions and their responsiveness. Detailed diagnostics provided by the test accurately pinpoint the exact transaction URLs that are slow/stalled, the total round-trip time of each transaction, and also indicate when such transaction requests were received by that node. The slowest transaction in the group can thus be identified.

For this test to run and report metrics on a WildFly JBoss server, you first need to BTM-enable the WildFly JBoss JVM. To know how, refer to the Installing eG Java BTM on JBoss WildFly topic in the *Java Business Transaction Monitoring* document.

Then, proceed to configure this test. Refer to the Java Business Transactions Test topic in the *Java Business Transaction Monitoring* document to know how to configure this test and learn about the metrics it reports.

About eG Innovations

eG Innovations provides intelligent performance management solutions that automate and dramatically accelerate the discovery, diagnosis, and resolution of IT performance issues in on-premises, cloud and hybrid environments. Where traditional monitoring tools often fail to provide insight into the performance drivers of business services and user experience, eG Innovations provides total performance visibility across every layer and every tier of the IT infrastructure that supports the business service chain. From desktops to applications, from servers to network and storage, from virtualization to cloud, eG Innovations helps companies proactively discover, instantly diagnose, and rapidly resolve even the most challenging performance and user experience issues.

eG Innovations is dedicated to helping businesses across the globe transform IT service delivery into a competitive advantage and a center for productivity, growth and profit. Many of the world's largest businesses use eG Enterprise to enhance IT service performance, increase operational efficiency, ensure IT effectiveness and deliver on the ROI promise of transformational IT investments across physical, virtual and cloud environments.

To learn more visit www.eginnovations.com.

Contact Us

For support queries, email support@eginnovations.com.

To contact eG Innovations sales team, email sales@eginnovations.com.