Developing Microservices with webMethods Microservices Runtime

Innovation Release

Version 10.2

April 2018

WEBMETHODS

This document applies to webMethods Microservices Runtime Version 10.2 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: WMIC-DG-102-20180417

Table of Contents

About this Guide

This guide provides information about administering webMethods Microservices Runtime.

Note: webMethods Microservices Runtime provides a superset of the functionality available in webMethods Integration Server. For information about administering and using features in webMethods Integration Server, see the *webMethods Integration Server Administrator's Guide*

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Getting Started with webMethods Microservices Runtime

What Are Microservices?

Microservices are independently deployable units of logic in which each microservice performs a single business function. Applications built in the microservices architectural style are developed as a suite of microservices.

Microservices can be implemented in various ways, including as a set of services or as event and channel definitions. Microservices can be distinguished from other types of services in the webMethods suite as follows:

Type of Service	Description
Integration Server service	Single function call, such as an operation on an interface or as part of an API (that is, a related set of Integration Server services).
Web service	Formal, WSDL-based grouping of operations, usable for SOA. Many of these can be hosted in a single runtime.
Microservice	Collection of operations implemented as services or as messages and message handlers, deployed in one or more related Integration Server packages.

What Is webMethods Microservices Runtime?

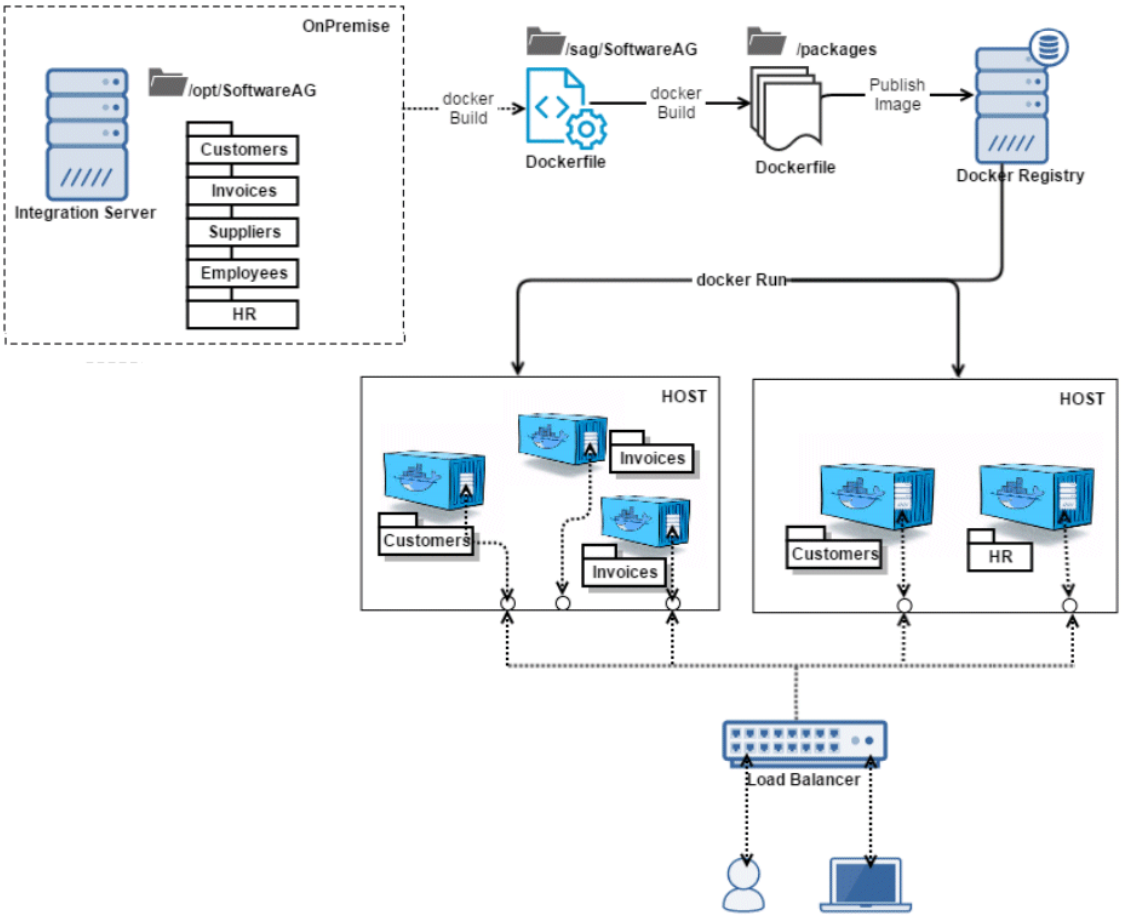
Software AG offers a lightweight container called webMethods Microservices Runtime to host microservices you develop in Software AG Designer. Using Microservices Runtime, you can deliver microservices as an Integration Server package that includes a set of related services, interfaces, document types, and triggers that subscribe to topics or queues, or as a set of related packages of this kind (for example, five packages relating to Human Resources functions).

Each microservice can run in its own Microservices Runtime and can communicate with lightweight mechanisms such as an HTTP resource API. However, you can also execute multiple microservices in the same Microservices Runtime. This hybrid solution enables you to separate microservices when needed, but also to group them when necessary. For example, suppose that you have two microservices that need to be scaled together in similar ways (that is, when you need a new instance of one, you need a new instance of the other). If you discover that one microservice is more heavily loaded than the other, or needs to be enhanced or updated more often, you could deploy the two microservices to separate Microservices Runtimes. If both microservices tend to be updated at the same time, you could cohost them in the same Microservices Runtime.

Microservices Runtime is fully compatible with webMethods Integration Server and can also host services you develop using Software AG Designer and Integration Server. While Microservices Runtime is optimized to have a reduced disk and memory footprint, you can convert it into a full Integration Server by installing additional modules, such as support for an external database.

Microservices Runtime provides out-of-the-box support for dynamic lookup of service endpoints using the open-source service registry named Consul. You can make your microservice available for remote access by registering the endpoint of the microservice container instance in the Consul service registry. Microservices Runtime provides services for automating the registration and deregistration process. Microservices Runtime provides facilities to look up the endpoint information which can be used to call the microservice at run time. You can also create your own package integrating with any other service registry provider. The package would provide services similar to those described for Consul in this guide.

Microservices Runtime is optimized for execution in a Docker container. You can run a microservice or a set of related microservices in a Docker container, obviating the need to purchase expensive virtual machines. Docker images include configuration, enabling you to deploy the exact same configuration anywhere. The Docker image can include one package or a set of related packages.



What Is Microservices Runtime Administrator?

Microservices Runtime Administrator is an HTML-based utility you use to administer Microservices Runtime. It allows you to monitor server activity, manage user accounts, make performance adjustments, and set operating parameters.

You can run the Microservices Runtime Administrator from any browser-equipped workstation on your network. Microservices Runtime Administrator is a browser-based application that uses services to accomplish its work.

2 Starting, Shutting Down, and Restarting Microservices Runtime

Starting Microservices Runtime and Microservices Runtime Administrator

Microservices Runtime must be running in order for clients to execute services or for Microservices Runtime to send outbound requests. If you are using Microservices Runtime in a development environment, it must be running in order for your developers to build, update, and test services using the Software AG Designer.

Before starting Microservices Runtime, make sure there is enough free disk space on the host machine to accommodate the storage of configuration and log files on disk. Running out of disk space can affect performance and lead to errors.

For information about starting Microservices Runtime from the command prompt or in safe mode, see the *webMethods Integration Server Administrator's Guide*.

Starting Microservices Runtime on Windows

To start Microservices Runtime on Windows

- Click **Start > All Programs > Software AG > Start Servers > Start Integration Server > *instance_name***

Starting Microservices Runtime on UNIX

To start Microservices Runtime on UNIX

1. Log in as a non-root user.

   Note: Running the script as root might reduce the security of your system.

2. Go to the *Integration Server_directory* /profiles/*instance_name* /bin directory and run the startup.sh script file.
3. If Microservices Runtime has been configured to request a master password for outbound password encryption, you will be prompted for this password in a popup window or from the server console. For information about managing outbound passwords, see the *webMethods Integration Server Administrator's Guide*.

Starting Microservices Runtime Administrator

To start Microservices Runtime Administrator

1. Do one of the following:
   - Open a browser and point it to the host and port where Microservices Runtime is running (for example, `http://localhost:5555` or `http://EXAMPLE:4040`).

- On Windows, click **Start > All Programs > Software AG > Administration > Integration Server Administrator > *instance_name***.

2. Log on to Microservices Runtime Administrator with a user name and password that has administrator privileges. User names and passwords are case sensitive.

Important: The default values are Administrator/manage. For security reasons, Software AG strongly recommends you change the user name and password from the default as soon as possible.

Shutting Down Microservices Runtime

When you shut down Microservices Runtime, all active sessions also shut down. for instructions on viewing active sessions before shutting down, see *webMethods Integration Server Administrator's Guide*.

Shutting Down Microservices Runtime on Windows

To shut down Microservices Runtime on Windows

- Click **Start > All Programs > Software AG > Stop Servers > Stop Integration Server > *instance_name***

Shutting Down Microservices Runtime from Microservices Runtime Administrator on Windows or UNIX

To shut down Microservices Runtime from Microservices Runtime Administrator on Windows or UNIX

1. In Microservices Runtime Administrator, in the upper right corner of any screen, click **Shutdown and Restart**.
2. Specify whether you want you want Microservices Runtime to wait before shutting down or shut down immediately.

To...	Do this
Shut down after *number* minutes or after all client sessions are complete	Select **After all client sessions end**. Then in the **Maximum wait time** field, specify the number of minutes you want Microservices Runtime to wait before restarting. Microservices Runtime will begin monitoring user activity and then automatically shut down after all non-administrator sessions complete or after the time you specify elapses, whichever comes first.

To...	Do this
Shut down immediately	Select **Immediately**. Microservices Runtime and all active sessions will terminate immediately.

3. Click **Shut Down**.

Restarting Microservices Runtime

You should restart Microservices Runtime when:

- **You make certain configuration changes.** Some configuration changes require Microservices Runtime to be restarted before they take effect. The documentation indicates when restart is necessary.
- **You want to incorporate updated services that cannot be dynamically reloaded.** This typically occurs for non-Java services.

To restart Microservices Runtime

1. In Microservices Runtime Administrator, in the upper right corner of any screen, click **Shutdown and Restart**.
2. Specify whether you want Microservices Runtime to wait before restarting or restart immediately.

To...	Do this
Restart after *number* minutes or after all client sessions are complete	Select **After all client sessions end**. Then in the **Maximum wait time** field, specify the number of minutes you want Microservices Runtime to wait before restarting. Microservices Runtime will begin monitoring user activity and then automatically restart after all non-administrator sessions complete or after the time you specify elapses, whichever comes first.
Restart immediately	Select **Immediately**. Microservices Runtime and all active sessions will terminate immediately. Then Microservices Runtime restarts.

3. Click **Restart**.

3 Configuring Microservices Runtime

Configure Microservices Runtime as you would configure Integration Server. See the *webMethods Integration Server Administrator's Guide* for instructions.

You might have installed these additional components when you installed Microservices Runtime:

Component	For more information . . .
CentraSite Asset Publisher Support	*webMethods Service Development Help*
Common Directory Service Support	*webMethods Integration Server Administrator's Guide*
Digital Event Services Support	*webMethods Integration Server Administrator's Guide*
Event Routing Support	*Communicating Between Software AG Products Using Event Routing*
External RDBMS Support	*webMethods Integration Server Administrator's Guide*

You can run a microservice or a set of related microservices in a Docker container, obviating the need to purchase expensive VMs. Docker images include configuration, enabling you to deploy the exact same configuration anywhere. The Docker image can include one package or a set of related packages. For instructions on using Docker, see *webMethods Integration Server Administrator's Guide*.

4 Using a Circuit Breaker with Services

About Circuit Breaker

Circuit breaker is an established design pattern that applications implement to prevent a failure in one part of the system from cascading to the rest of the system. In an architecture with distributed applications, such as microservices, many services call other services running on remote servers. If the remote service is unavailable or the network is slow, the calling service may wait until a timeout occurs. During this time, the calling service continues to consume critical resources such as server threads and memory. If multiple services call the unresponsive or failing remote service, the impact of the remote service cascades throughout all the calling services, causing even more resources to be consumed and affected by a single failing service. Implementing a circuit breaker on the call to the remote service can prevent the impact of the failing or unresponsive service or network latency from cascading throughout the system.

The circuit breaker design pattern works much like an electrical circuit breaker which is intended to “trip” or open the circuit when failure is detected. This prevents the flow of electrical current through the circuit. After a time delay, the electrical circuit breaker resets and closes the circuit, which causes the flow of electricity to resume.

In a software application, a circuit breaker functions as a proxy that executes the remote services and monitors the remote service for failures. A failure can be an exception and/or a timeout. When the number of failures meets a predetermined threshold within a specified time period, the circuit breaker “trips” or opens the circuit. Subsequent requests for the service end with an error or result in execution of an alternative service. After a reset period elapses, the circuit breaker sets the circuit state to half-open and executes the next request for the service. By allowing a single request to execute and causing other requests to wait, the circuit breaker gauges the health of the service. Upon success of the service, the circuit breaker closes circuit and waiting requests proceed. However, if the service ends with another failure, the circuit breaker re-opens the circuit.

Circuit breakers can be especially useful in systems with a microservices architecture as these systems often feature a large number of distributed components. By configuring a circuit breaker on the invocation of the remote service, you can limit the impact the abnormal behavior of a remote service on other micorservices and critical resources in your system.

Note: The circuit breaker feature is available by default for a service that resides in a Microservices Runtime. To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing.

Circuit States

A circuit for a service can have the following states which determine how circuit breaker responds to request to invoke the service:

If the circuit state is...	Then...
Closed	The circuit breaker executes the service.
Open	The circuit breaker does not execute the service. Instead, the circuit breaker returns an exception, which allows the request to fail immediately, or circuit breaker invokes an alternative service.
Half-open	The circuit breaker executes the service the next time it is requested. If the service executes successfully, then the circuit breaker changes the circuit state to closed. If the service ends because of a failure event (exception and/or timeout), the circuit breaker changes the circuit state to open. While the circuit is in a half-open state and circuit breaker is already executing a request for the service, any other requests for the service wait until the circuit exits the half-open state. If service execution is successful, circuit breaker closes the circuit and executes the waiting requests for the service.

How Does a Circuit Breaker for a Service Work?

A circuit breaker works by monitoring a service for failures. The circuit breaker allows service execution to proceed when failure events do not meet an established threshold. However, when the number of failure events meets the established failure threshold within the failure time period, the circuit breaker opens the circuit, preventing further execution of the service. The following table provides an overview of how a circuit breaker works.

Step	Description
1	The server receives a request to invoke a service. Upon receiving a request to invoke a service, the server first determines whether or not a circuit breaker is configured for the service. ■ If a circuit breaker is configured for the service, the server passes the request to the circuit breaker which checks the state of the circuit as described in step 2, below. ■ If a circuit breaker is not used with the service, the server invokes the service.

Step	Description
2	The circuit breaker examines the state of the circuit for the service. ■ If the circuit is closed, the circuit breaker invokes the service using a new thread from the circuit breaker thread pool, which is separate from the server thread pool. The thread that called the service originally waits for the results of the service execution. ■ If the circuit is open, the circuit breaker does not invoke the service. See step 5, below, for more information about how the circuit breaker responds to a request for service with an open circuit. ■ If the circuit is half-open and this is the first request for the service since the circuit state became half-open, the circuit breaker invokes the service. When executing a service in a half-open state, any other requests for the service wait until the circuit exits the half-open state. For more information, see step 8, below.
3	When the circuit state is closed, upon execution of the service by the circuit breaker, one of the following occurs: ■ The service executes successfully. **Note:** If the **Failure event** property is set to **Timeout only** or **Exception or timeout**, a successful execution indicates that the service executed successfully within the specified timeout period. ■ The service ends with an exception. If the **Failure event** property is set to **Exception only** or **Exception or Timeout**, the circuit breaker considers the exception to be a failure event. The circuit breaker increments the failure count for the service by 1. ■ The service does not execute to completion before the timeout period elapses. If the **Failure event** property is set to **Timeout only** or **Exception or Timeout**, the circuit breaker considers the timeout to be a failure event. The circuit breaker increments the failure count for the service by 1. If the **Cancel thread on timeout** property is set to true, the circuit breaker attempts to cancel the thread executing the service. If the service ends with a failure event and it is the first failure event, the circuit breaker starts the failure timeout period. If the

Step	Description
	number of failure events specified in **Failure threshold** property occurs before the failure timeout period ends, the circuit opens. Circuit breaker determines the time that the failure period ends by adding the number of seconds specified in the **Failure period** property to the time the first failure event occurred. If the service executes successfully and completes after the failure timeout period ends, circuit breaker resets the failure count for the service to 0.
4	Subsequent invocations of the service result in the circuit breaker opening or "tripping" the circuit for the service if: ■ The number of failure events for the service equals the **Failure threshold** property value *and* ■ The failure events occur within the failure timeout period which circuit breaker determines by adding the value of **Failure period** property to the time of the first failure event. The circuit breaker starts the reset period which determines the length of time to keep the circuit in an open state. The **Circuit reset period** property determines the length of the reset period. For example, suppose that circuit breaker treats exceptions as a failure event, the **Failure threshold** property is set to 5, and the **Failure period** is set to 60 seconds. Further suppose that the first failure event occurs at 10:07:10. Circuit breaker starts the failure timeout period. The failure timeout period ends at 10:08:10 which circuit breaker determines by adding the **Failure period** value to the time of the first failure event. If 5 invocations of the service end with an exception in less than 60 seconds, that is, before 10:08:10, the circuit breaker opens the circuit for the service.
5	When the circuit breaker receives a request to execute the service, the circuit breaker does one of the following: ■ If the **Circuit open action** property is set to **Throw exception**, the circuit breaker responds to the invoke request by throwing the exception that caused the circuit to open. ■ If the **Circuit open action** property is set to **Invoke service**, the circuit breaker executes the alternate service specified in the **Circuit open service** property and returns the results to the calling service. The circuit breaker places the *$circuitBreakerService* and *$circuitBreakerEvent* parameters in the input pipeline for the alternate service. For more information about the circuit open

Step	Description
	service, see "Building a Service for Use with an Open Circuit" on page 23. **Note:** When the circuit is open, circuit breaker does not use the circuit breaker thread pool to throw the exception or execute the alternate, open circuit service. Instead, circuit breaker uses the same thread that executes the calling service to return the exception or execute the alternate, open circuit service.
6	The circuit breaker repeats the previous step in response to a request for the service until the time specified in the **Circuit reset period** property elapses.
7	When the circuit reset period elapses, the circuit breaker sets the circuit to a half-open state.

8	The circuit breaker receives a request for the service.
	■ If this is the first request for the service since the circuit state became half-open, the circuit breaker invokes the requested service. One of the following occurs:
	  ■ If the service executes successfully, the circuit breaker closes the circuit. The circuit breaker will invoke the service upon subsequent service executions. The circuit breaker resets the failure count to 0 (zero).
	  ■ If service execution results in a failure event, the circuit breaker re-opens the circuit and restarts the reset period. The circuit breaker proceeds as described in step 5.
	■ While the circuit is in a half-open state and circuit breaker is already executing a request for the service, any other requests for the service wait until the circuit exits the half-open state. If service execution is successful, circuit breaker closes the circuit and executes the waiting requests for the service. If service execution is not successful, circuit breaker re-opens the circuit and responds to the waiting services as described in step 5.

Configuring a Circuit Breaker for a Service

You can configure a circuit breaker for any user-defined service. Use the Service Development perspective in Software AG Designer to enable and configure a circuit

breaker for a service. For configuration instructions, considerations, and guidelines, see the *webMethods Service Development Help*.

Building a Service for Use with an Open Circuit

When a circuit for a service is open, the service does not execute. Instead, circuit breaker responds to requests for the service by throwing the same exception that caused the last failure event or by executing an alternate service. The alternate service, known as the *open circuit service* , is a user-defined service that performs an action that makes sense for your application. In some applications, you might want the open circuit service to return a cached value or an alternate result to the calling service. You can also code the open circuit service to perform a different action based on whether it is the first invocation of the open circuit service since the circuit opened or if it is a subsequent invocation.

Whether a circuit breaker throws an exception or executes an alternate service depends on the value of the **Circuit open action** property. If you specify **Invoke service** as the circuit open action, you must identify the service to invoke in the **Circuit open service** property.

When circuit breaker invokes the open circuit service, circuit breaker places the following parameters in the service pipeline for the open circuit service.

Parameter Name	Description
$circuitBreakerService	**String.** Fully qualified name of the service with the open circuit.
$circuitBreakerEvent	**String.** Indicates whether this is the first time circuit breaker called the open circuit service since the circuit opened or if this is a subsequent invocation. The *$circuitBreakerEvent* parameter has one of the following values: ■ `CIRCUIT_OPENED` if this is the first execution of this open circuit service since the circuit opened. ■ `CIRCUIT_OPEN` if this is not the first execution of the open circuit service since the circuit opened.

Note: The open circuit service has access to the above parameters as well as any parameters that existed in the pipeline at the time of the execution request for the service with a configured circuit breaker.

Keep the following information in mind when building an open circuit service for use with a circuit breaker:

- If the open circuit service interacts with a remote resource, such as a database or web server, make the interaction asynchronous to prevent a service execution from blocking other threads or delaying the execution of the original calling service.

- An open circuit service can be used with more than one service with a configured circuit breaker.
- Circuit breaker does not use a thread from the circuit breaker thread pool to execute the open circuit service. Circuit breaker uses the same thread that executed the calling service to execute the open circuit service.
- If an exception occurs while executing this service, it does not impact the circuit breaker failure count for the originally requested service.
- Software AG recommends that the open circuit service is not configured to use a circuit breaker.
- Software AG recommends that the open circuit service is not the same as the service with the configured circuit breaker. That is, do not create a circular situation where a service with an open circuit calls itself.

Configuring the Circuit Breaker Thread Pool

Circuit breaker uses a dedicated thread pool, separate from the server thread pool, to execute services for which a circuit breaker is configured. This thread pool is referred to as the circuit breaker thread pool. You can specify the minimum and maximum number of threads in the circuit breaker thread pool.

At run time, circuit breaker uses a thread from the circuit breaker thread pool to execute the requested service, passing the service invocation pipeline to the new thread. This thread is separate from the thread executing the calling service. The calling service waits for a response from the requested service. Circuit breaker returns the service results and then returns the thread to the circuit breaker thread pool. The calling service then proceeds with execution. If the requested service is configured to treat a time out as a failure event and the service does not execute to completion before the timeout period elapses, circuit breaker returns an exception to the calling service. If the **Cancel thread on timeout** property is set to false, circuit breaker orphans the thread. If the **Cancel thread on timeout** property is set to true, circuit breaker attempts to cancels the thread. If the thread cannot be canceled, circuit breaker abandons the thread.

Circuit breaker uses a separate thread pool because it decouples the thread that executes the calling service from the thread that executes the requested service. This decoupling allows the calling service to proceed with execution if the requested service does not complete before the timeout period elapses. As a result, failures can return quickly. For example, suppose that a circuit breaker is configured for a service that reads information from a database. If the database goes off line, an attempt to connect to the unavailable database and execute a query may wait a while before returning because network input/output operations typically cannot be interrupted. By using a separate thread for the requested service, the circuit breaker can abandon the thread and return an exception to the client without needing to wait for the input/output operation to complete.

Note: Circuit breaker uses the circuit breaker thread pool to execute only those services for which a circuit breaker is configured. Circuit breaker does not

use a thread from the circuit breaker thread pool to execute the circuit open service.

You can configure the size of the circuit breaker thread pool by specifying the minimum and maximum number of threads for the pool. When the server starts, the circuit breaker thread pool initially contains the minimum number of threads. The server adds threads to the pool, as needed, until the pool contains the maximum number of allowed threads. If the pool reaches the maximum number of threads, before executing the next requested service with a configured circuit breaker, circuit breaker must wait for a thread to be returned to the pool.

The server provides server configuration parameters for specifying the minimum and maximum number of threads in the circuit breaker thread pool.

- watt.server.circuitBreaker.threadPoolMin specifies the minimum number of threads that the server maintains in the circuit breaker thread pool. The circuit breaker thread pool is used to execute services with a configured circuit breaker. When the server starts, the circuit breaker thread pool initially contains this minimum number of threads. The server adds threads to the pool as needed until it reaches the maximum allowed, which is specified by the watt.server.circuitBreaker.threadPoolMax. You must specify a value greater than or equal to 0 (zero) and less than or equal to the value of watt.server.circuitBreaker.threadPoolMax. The default is 10.

  Note: You must restart Microservices Runtime for changes to take effect.

- watt.server.circuitBreaker.threadPoolMax specifies the maximum number of threads that the server maintains in the circuit breaker thread pool. The circuit breaker thread pool is used to execute services with a configured circuit breaker. If this maximum number is reached, the server waits until services complete and return threads to the circuit breaker thread pool before running more services with a configured circuit breaker. You must specify a value greater than 0 and greater than or equal to the value of watt.server.circuitBreaker.threadPoolMin. The default is 75.

  Note: You must restart Microservices Runtime for changes to take effect.

Use the server configuration properties to size the circuit breaker thread pool appropriately for your environment. Keep in mind that all services for which a circuit breaker is configured share the circuit breaker thread pool.

Circuit Breaker Statistics

Microservices Runtime gathers circuit breaker statistics for each service with a configured circuit breaker. Microservices Runtime Administrator displays statistics in the **Circuit Breaker Information** table on the **Server > Service Usage** page. Microservices Runtime maintains the following circuit breaker information for each service with a configured circuit breaker.

Field	Description
Name	Name of the service for which a circuit breaker is configured.
State	The state of the circuit. The circuit state can be one of the following: ■ Closed. The service executes. ■ Open. The service does not execute. Instead, depending on the circuit breaker configuration, the service returns an exception or executes an alternative service. ■ Half-Open. The first request for this service since the circuit state changes to half-open results in service execution. All other requests wait. If the service executes successfully, the circuit is closed and waiting requests execute. If the service ends with a failure exception, the circuit is re-opened. For a detailed explanation of possible circuit states, see the "Circuit States" on page 18.
Open Time	Time at which circuit breaker last set the circuit state to open.
Half-Open Time	Time at which circuit breaker last set the circuit state to half-open.
Closed Time	Time at which the circuit breaker last set the circuit state to closed.
Open Count	Number of times that circuit breaker set the circuit state to open since Microservices Runtime started.
Request Count	Number of incoming requests for the service since the circuit breaker changed the circuit state to open. For information about how circuit breaker handles requests for a service with an open circuit, see "How Does a Circuit Breaker for a Service Work?" on page 19.

Note: If a circuit breaker is not configured for any service, the **Circuit Breaker Information** table displays the message "No Services with a Circuit Breaker Enabled".

5 Consul Support

Configuring Connections to Consul Server

Configure one or more server aliases for the public services provided in WmConsul to use to connect to a Consul server. You must create at least one server alias for the services to execute successfully. An alias name used as the *registryAlias* input parameter value for a WmConsul public service must exist in the Microservice Consul Registry Server alias list.

Note: The WmConsul package contains the public services for interacting with a Consul server.

To create an alias to the Consul server

1. Open Microservices Runtime Administrator.
2. in the **Packages** menu of the Navigation panel, click **Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, click **Create Microservice Consul Registry Server Alias**.
5. Under Microservice Consul Registry Server Alias Properties, provide the following information.

For this field...	Specify...
Alias	Name for the server alias.
Host Name or IP Address	Location of the Consul server as a host name or IP address.
Port Number	Port on which to connect to the Consul server.
Enable Consul Health Check	Whether to enable a health check for microservices registered using this alias. Consul uses the health check to keep track of the health of a registered microservice . Select **Yes** to enable health checks.
User Name	Optional. If the Consul server is configured to use a user name and password, the user name.
Password	Optional. If the Consul server is configured to use a user name and password, the password.

For this field...	Specify...
Use SSL	Whether to use a secure connection to connect to the Consul server. To use SSL, select **Yes** and then provide truststore and possibly keystore information. **Note:** If you select yes, the Consul server must be configured to accept HTTPS requests.
Keystore Alias	Optional. Keystore alias that contains the client certificates to use for the secure connection. You need to provide this information only if **Use SSL** is set to Yes and the registry server requires client certificates.
Key Alias	Optional. Key alias for the private key and certificates to use for establishing a secure connection. You need to provide this information only if **Use SSL** is set to **Yes** and the Consul server requires client certificates .
Truststore Alias	Optional. Truststore alias that contains the Consul server's certificate authority certificates. You need to provide this information only if **Use SSL** is set to Yes.

6. Click **Save Changes**.

Testing an Alias for the Consul Server

After you add an alias for a Consul server, you can test the alias to ensure that Microservices Runtime can establish a connection to the Consul server using the information provided in the alias.

To test a Consul server alias

1. Open Microservices Runtime Administrator.
2. in the **Packages** menu of the Navigation panel, click **Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, in the Microservce Consul Registry Server List click in the Test column for the alias you want to test.

   Microservices Runtime Administrator displays a status message above the list of aliases that indicates whether or not the connection is successful.

Setting the Default Alias for the Consul Server

You can identify one of the Consul server aliases as the default alias. The pub.consul.client services will use this alias to connect to the Consul server if you do not specify a different alias in the *registryAlias* input parameter.

To specify the default Consul server alias

1. Open Microservices Runtime Administrator.
2. in the **Packages** menu of the Navigation panel, click **Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, click **Change Default Alias**.
5. On the Microservice Consul Registry Servers > Change Default Alias page, in the **Default Alias** list, select the Consul server alias to use as the default.
6. Click **Update**.

Deleting a Consul Server Alias

If you no longer need an alias to a Consul server, you can delete it.

To delete a Consul server alias

1. Open Microservices Runtime Administrator.
2. in the **Packages** menu of the Navigation panel, click **Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, in the Microservice Consul Registry Server List click ✕ in the **Delete** column for the alias you want to delete. Microservices Runtime Administrator prompts you to confirm deleting the alias. Click **OK**.

   If you delete the default Consul server alias, Microservices Runtime Administrator displays a status message stating there is no longer a default Consul server alias which can prevent microservice registration.

Consul Public Services Folder

The following elements are available in this folder:

Element	Package and Description
pub.consul.client:deregisterService	WmConsul. De-registers a microservice from a Consul server. Use as a shutdown service for the package being registered as a microservice.
pub.consul.client:getAllHostsForService	WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul.
pub.consul.client:getAnyHostForService	WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul and if there are multiple hosts for this microservice, randomly return one of those hosts.
pub.consul.client:registerService	WmConsul. Registers a microservice with a Consul server. Use as a startup service for the package being registered as a microservice.

pub.consul.client:deregisterService

WmConsul. De-registers a microservice from a Consul server. Use this service as a shutdown service for the package being registered as a microservice.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice name to de-register from the Consul server.

Output Parameters

None

Usage Notes

A package that identifies pub.consul.client:deregisterService as a shutdown service must identify WmConsul as a package dependency.

pub.consul.client:getAllHostsForService

WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice for which you want to retrieve a list of active hosts.

Output Parameters

hostList	String List. An array containing the names of hosts of the microservice. Each element is in the format *host* :*port* (for example, appserver.xyz.com:4555). If there are no hosts registered for the given microservice, returns null.

pub.consul.client:getAnyHostForService

WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul and if there are multiple hosts for this microservice, randomly return one of those hosts.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice for which you want to retrieve an active host.

Output Parameters

hostName	String. Name of a host of the microservice, chosen randomly. The host name is in the format: *port* (for example,

appserver.xyz.com:4555). If there are no hosts registered for the given microservice, returns null.

pub.consul.client:registerService

WmConsul. Registers a microservice with a Consul server. Use this service as a startup service for the package being registered as a microservice.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias.
microserviceName	String. Microservice name to register with the Consul server.

Output Parameters

None

Usage Notes

A package that identifies pub.consul.client:registerService as a startup service must identify WmConsul as a package dependency.