Developing Microservices with webMethods Microservices Container

Version 10.1

October 2017

WEBMETHODS

This document applies to webMethods Microservices Container Version 10.1 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-101-20180822

Table of Contents

About this Guide

This guide provides information about administering webMethods Microservices Container.

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 Container

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 Container?

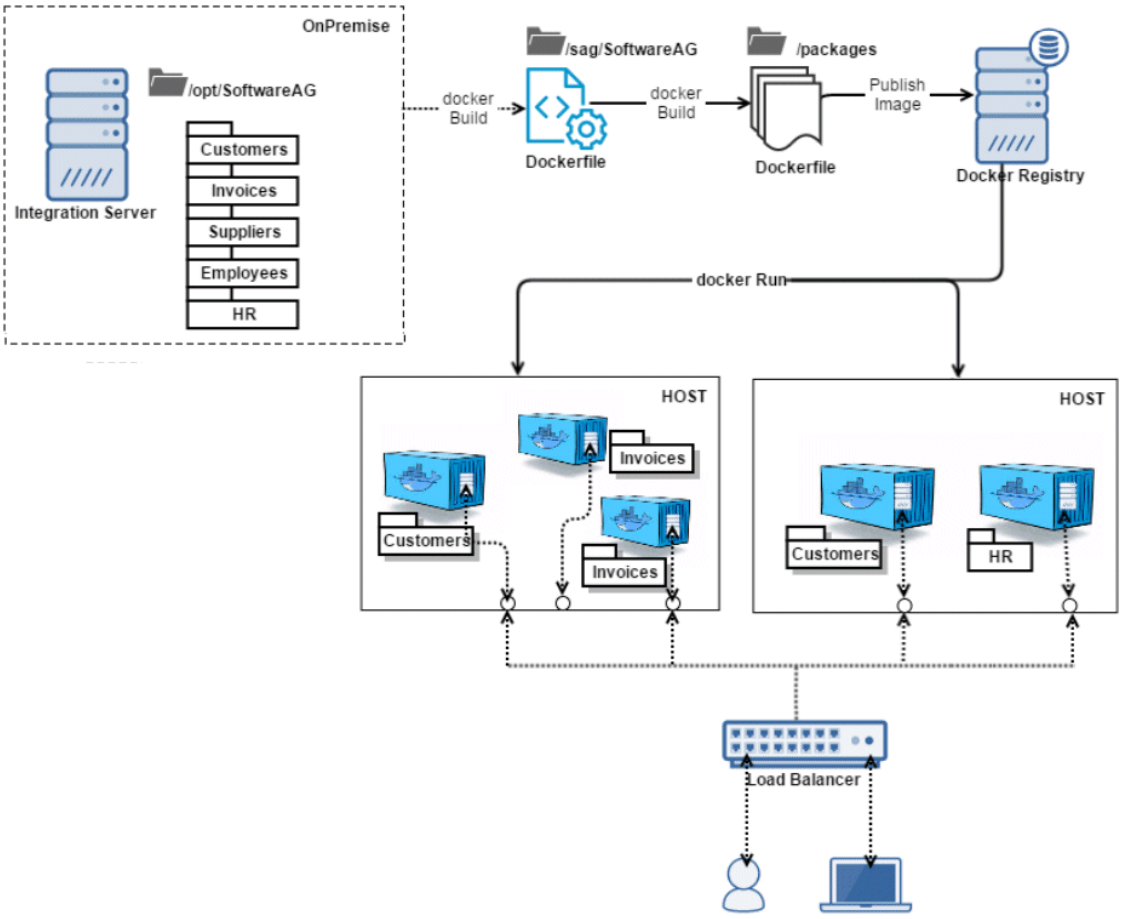
Software AG offers a lightweight container called webMethods Microservices Container to host microservices you develop in Software AG Designer. Using Microservices Container, you can deliver microservices as an Integration Server package that includes a set of related services, interfaces, document types, 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 Container and can communicate with lightweight mechanisms such as an HTTP resource API. However, you can also execute multiple microservices in the same Microservices Container. 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 Containers. If both microservices tend to be updated at the same time, you could cohost them in the same Microservices Container.

Microservices Container is fully compatible with webMethods Integration Server and can also host services you develop using Software AG Designer and Integration Server. While Microservices Container 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 Container 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 Container provides services for automating the registration and deregistration process. Microservices Container 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 Container 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 Container Administrator?

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

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

2 Starting, Shutting Down, and Restarting Microservices Container

Starting Microservices Container and Microservices Container Administrator

Microservices Container must be running in order for clients to execute services or for Microservices Container to send outbound requests. If you are using Microservices Container 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 Container, 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 Container from the command prompt or in safe mode, see the *webMethods Integration Server Administrator's Guide*.

Starting Microservices Container on Windows

To start Microservices Container on Windows

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

Starting Microservices Container on UNIX

To start Microservices Container 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 Container 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 Container Administrator

To start Microservices Container Administrator

1. Do one of the following:
   - Open a browser and point it to the host and port where Microservices Container 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 Container 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 Container

When you shut down Microservices Container, 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 Container on Windows

To shut down Microservices Container on Windows

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

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

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

1. In Microservices Container Administrator, in the upper right corner of any screen, click **Shutdown and Restart**.
2. Specify whether you want you want Microservices Container 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 Container to wait before restarting. Microservices Container 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 Container and all active sessions will terminate immediately.

3. Click **Shut Down**.

Restarting Microservices Container

You should restart Microservices Container when:

- **You make certain configuration changes.** Some configuration changes require Microservices Container 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 Container

1. In Microservices Container Administrator, in the upper right corner of any screen, click **Shutdown and Restart**.
2. Specify whether you want Microservices Container 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 Container to wait before restarting. Microservices Container 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 Container and all active sessions will terminate immediately. Then Microservices Container restarts.

3. Click **Restart**.

3 Configuring Microservices Container

Configure Microservices Container 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 Container:

Component	For more information . . .
CentraSite Asset Publisher Support	*webMethods BPM and CAF CentraSite Metadata Help*
Common Directory Service Support	*webMethods Integration Server Administrator's Guide*
Developer Support	*webMethods Integration Server Administrator's Guide*
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 Container. To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing.

Note: To use the circuit breaker functionality in version 10.1, you must install the following fixes: ESB_10.1_Fix2 and IS_10.1_Core_Fix2.

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.

Step	Description
	■ 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.
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

Step	Description
	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 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.

Step	Description
	■ 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 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 Container 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 Container 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 Container gathers circuit breaker statistics for each service with a configured circuit breaker. Microservices Container Administrator displays statistics in the **Circuit Breaker Information** table on the **Server > Service Usage** page. Microservices Container 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 19.
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 Container 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

Field	Description
	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 Using Configuration Variables Templates with Microservices Container

About Configuration Variables Templates

Microservices Container provides the ability to create a Docker image from an installed and configured instance of Microservices Container. The Docker image contains the Microservices Container application including packages and Microservices Container assets such as ports, JMS connection aliases, keystores, and JDBC pools. Because you can execute multiple Docker containers from a single Docker image, a Docker image for a Microservices Container instance might be used across development, testing, and production environments. While the microservices and accompanying artifacts are the same across the running containers and stages, configuration information might be different. For example, user name and password combinations, proxy server host and port, remote server host and port, and global variable values might be different across different instances of running Docker containers. While a Microservices Container stores configuration information inside its file system and therefore inside any Docker image created from the Microservices Container, some configuration information can be externalized and passed to the Microservices Container at startup. With Microservices Container you can accomplish this through use of a configuration variables template and environment variables.

A configuration variables template contains configuration properties that map to properties on the Microservices Container. The property values can be set externally in the template and then passed to a Microservices Container when it starts up. As part of the startup process, Microservices Container loads the information from the configuration variables template and replaces the configuration information stored in the file system.

By externalizing configuration information, a single Docker image created for a Microservices Container can be used across multiple environments, including different stages of the production cycle. For example, you might use different templates for specific environments such as testing versus production. Or you might use the same template for all environments but use environment variables to vary the configuration in each environment.

While the primary purpose of configuration variables is to extend the usefulness of single Microservices Container image by making it possible to reuse a single Docker image across multiple stages in the production cycle, you can also use configuration variables templates in situations where other deployment options are too heavyweight.

Note: The configuration variables feature is available by default for Microservices Container. To use the configuration variables feature with Integration Server, your Integration Server must have additional licensing.

Note: To use configuration variables template in version 10.1, you must install IS_10.1_Core_Fix7 which includes PIE-52236, the feature that provides support for configuration variables.

What Does a Configuration Variables Template Look Like?

A configuration variables template is a properties file that contains configuration data as a series of key-value pairs where the key name reflects the asset and particular asset property for which you can supply a value. For example, `jndi.DEFAULT_IS_JNDI_PROVIDER.providerURL=nsp://myHost:9000` indicates that the JNDI provider alias named DEFAULT_IS_JNDI_PROVIDER has a URL of nsp://myHost:9000.

When a configuration variables template is generated from an existing Microservices Container, the template is populated with configuration data from the Microservices Container instance. The template content reflects the Microservices Container configuration at the time Microservices Container generated the template. For example, the configuration variables template lists a key-value pair for each defined global variable.

The generated configuration variables template does not contain key-value pairs for all of configuration information for Microservices Container. This is because only a subset of configuration information is supported for use in configuration variables template.

Below is an excerpt of a configuration variables template:

```
#Sample Generated Template
#Wed Jun 20 17:13:40 EDT 2018
email.WmRoot.myEmailPort.host=exchange
email.WmRoot.myEmailPort.password={AES}VphDydln9QyvLIGf+FASxw\=\=
email.WmRoot.myEmailPort.server_port=7891
email.WmRoot.myEmailPort.user=user123ispasswordexpirationsettings.
jms.DEFAULT_IS_JMS_CONNECTION.clientID=DEFAULT_IS_JMS_CLIENT
jndi.DEFAULT_IS_JNDI_PROVIDER.providerURL=nsp\://localhost\:9000
settings.watt.server.compile=C\:\\IS_10-3\\June6\\jvm\\jvm\\bin\\javac
-classpath {0} -d {1} {2}
settings.watt.ssh.jsch.ciphers=aes256-ctr,aes192-
ctr,arcfour,arcfour128,arcfour256,aes128-ctr,aes128-cbc,3des-ctr,3des
-cbc,blowfish-cbc,aes192-cbc,aes256-cbc
```

Only specific values can be supplied via the template for an asset. For example, for a JMS connection alias a configuration variables template lists a key-value pair for the clientID, user, and password properties. The template omits key-value pairs for all other properties.

In the template, each property name follows this pattern: *`assetType.assetName.propertyName=value`* where *`assetType`* is the type of administrative asset, *`assetName`* is the name given to the asset such as an alias name, and *`propertyName`* is the asset property for which you can set a value.

For example, following are key-value pairs for a global variable named `serverhost`, a global variable named `serverport`, the DEFAULT_IS_JMS_CONNECTION connection alias, and the JDBC pool alias named webMPool:

```
globalvariable.serverhost.value=server123.example.com
globalvariable.serverport.value=5555
jms.DEFAULT_IS_JMS_CONNECTION.clientID=DEFAULT_IS_JMS_CLIENT
jms.DEFAULT_IS_JMS_CONNECTION.password=encryptedValue
```

```
jms.DEFAULT_IS_JMS_CONNECTION.user= userA
jdbc.webMPool.dbURL=jdbc:wm:oracle://testserver:1521;serviceName=webm
jdbc.webMPool.password={AES}CszC/Yl1w1XqEK0B17RFucP0kgMHNt823RUU6ad39tw\=jdbc.we
bMPool.userid=jdbcuserTest
```

To change the assigned values in a configuration variables template, use a text editor to open the template and make changes. Any value you specify for a property must adhere to the requirements for that property. For example, the global variable value cannot exceed 255 characters. For passwords or other types of values that should be encrypted, Microservices Container Administrator provides a utility to encrypt a value. Microservices Container Administrator uses password handles and the Password-Based Encryption technology installed with Microservices Container for encryption.

As an alternative to "hard coding" a value for a property in a template, you can specify an environment variable (ENV variable) as the value of a property in the configuration variables template. Using ENV variables can make the configuration variables template more flexible. For example, the following line specifies that the JDBC pool URL for the alias webMPool should be set to the value of the environment variable named `JDBC_URL`:
`jdbc.webMPool.dbURL=$env{JDBC_URL}`

If you opt to use an ENV variable as a value for configuration variables, make sure that you specify the values for ENV variables defined in the configuration variables template. Refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers.

When Is the Template Applied?

Microservices Container applies a configuration variables template at startup. When Microservices Container starts, either in a Docker container or on-premises, Microservices Container looks for a configuration variables template in the following locations in the specified order.

1. The location identified by the `SAG_IS_CONFIG_PROPERTIES` environment variable.
2. *Integration Server_directory*/instances/*instanceName* /application.properties
3. *Integration Server_directory*/application.properties

Once Microservices Container locates a template, Microservices Container does not look for a template in any remaining locations.

Note: If Microservices Container does not find a configuration variables template at any of the above locations, Microservices Container proceeds with start up and does not apply a template.

Microservices Container processes the template, updating the configuration information in Microservices Container with the information in the template. While processing the template, Microservices Container writes log messages to the logs/configurationvariables.log to reflect the progress of the variable substitution. When substitution is complete, Microservices Container proceeds with startup.

While substituting variable values, Microservices Container ignores properties and values that do not correspond to existing assets on the Microservices Container. A configuration variables template can be used to change the configuration information only for assets that already exist on Microservices Container. That is, you cannot add assets to a Microservices Container by adding key-value pairs to a template

During substitution, Microservices Container ignores any values for properties that are not set on the Microservices Container. For example, suppose the configuration variables template file specifies the following for a JMS connection alias named `myAlias`:

```
jms.myAlias.clientID=abc
jms.myAlias.password=encryptedValue
jms.myAlias.user= userA
```

If the Microservices Container contains a JMS connection alias named “myAlias” and that alias specifies a **Connection Client ID** value of “xyz”, at startup, Microservices Container substitutes “abc” for “xyz”. If the myAlias connection alias does not specify values for **Username** or **Password**, Microservices Container does not substitute the values set in `jms.myAlias.password` or `jms.myAlias.user`.

Approaches for Using a Configuration Variables Template with Microservices Container

Because a configuration variables template can be used with a Microservices Container running in a Docker container or an on-premises Microservices Container, there are multiple approaches to using the template. The approach you select determines the template file name, the template file location, whether the template file is part of a Docker image, and how to start the Microservices Container so that Microservices Container applies the template.

To help determine which approach to use, decide the following:

- Will the Microservices Container with which you want to use the template run in a Docker container or on-premises (outside of a Docker contaier)?
- If Microservices Container runs in a Docker container, do you want to include the template as part of the Docker image or external to the image?
- Do you want to hard code values for variables in the template or use environment (ENV) variables to specify some or all values?

The answers to the above questions help determine which approach to use, which in turn, determines the files that you need to create, where you place the configuration variables template, the name of the file, and how to start the Microservices Container so that Microservices Container applies the template values.

The following table describes the possible approaches for using a configuration variables template with Microservices Container

Approach	Description and requirements
Docker image does not include template which does not use ENV variables	Microservices Container runs in a Docker container, the template is not included in the Docker image, and configuration variables values are hard coded in the template only. The template does not use ENV variables. Template file name: application.properties or a user-defined name Template location: A location that is accessible to the Docker container. Start up: The `docker run` command includes the `SAG_IS_CONFIG_PROPERTIES` environment variable which specifies the name and location of the configuration variables template.
Docker image does not include template and template uses ENV variables	Microservices Container runs in a Docker container, the template is not included in the Docker image, and configuration variable values are hard coded in the template and/or the template uses environment variables. Template file name: application.properties or a user-defined name Template location: A location that is accessible to the Docker container. If you are using ENV variables for some or all of the variable values, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers. Start up: The `docker run` command includes the `SAG_IS_CONFIG_PROPERTIES` environment variable which specifies the name and location of the configuration variables template. If using the `docker run` command, the command must also include options `--env, -e` or `--env-file` options for setting the ENV variables in the container.
Docker image includes template which uses ENV variables	Microservices Container runs in a Docker container, the configuration variables template is part of the Docker image, and the template uses environment variables for some or all the values in the template. Template name: application.properties

Approach	Description and requirements
	Template location is one of the following: ■ *Integration Server_directory*/instances/*instanceName* / application.properties ■ *Integration Server_directory*/application.properties If you are using ENV variables for some or all of the variable values, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers. Start up: The `docker run` command includes the `SAG_IS_CONFIG_PROPERTIES` environment variable which specifies the name and location of the configuration variables template. The command must also include options `--env, -e` or `--env-file` options for setting the ENV variables in the container.
On-premises Microservices Container installation does not include template and template does not use ENV variables.	Microservices Container runs on-premises, configuration variables values are hard coded in the template only, and the template is not included in the Microservices Container file system. The template does not use ENV variables. Template name: application.properties or a user-defined name Template location: A location that is accessible to the Microservices Container. Start up: Microservices Container start up includes the `SAG_IS_CONFIG_PROPERTIES` ENV variable which specifies the name and location of the configuration variables template.
On-premises Microservices Container includes template which does not use ENV variables.	Microservices Container runs on-premises and a configuration variables template defines all the values. The template does not use ENV variables. Template name: application.properties Template location is one of the following: ■ *Integration Server_directory*/instances/*instanceName* / application.properties ■ *Integration Server_directory*/application.properties

Approach	Description and requirements
	Start up: This approach does not require any special steps at start up. For information about starting Microservices Container, see "Starting, Shutting Down, and Restarting Microservices Container " on page 11.
On-premises Microservices Container includes template which uses ENV variables	Microservices Container runs on-premises and a configuration variables template hard codes the values and/or uses environment variables for the values. Template name: application.properties Template location is one of the following: ■ *Integration Server_directory*?/instances/*instanceName* /application.properties ■ *Integration Server_directory*/application.properties Environment variables can be specified in *Integration Server_directory*profiles/IS_*instanceName* /bin/custom_setenv.bat/sh files or ENV variables can be pre-defined in your environment. Start up: This approach does not require any special steps at start up. For information about starting Microservices Container, see "Starting, Shutting Down, and Restarting Microservices Container " on page 11.

Overview of Building a Configuration Variables Template

Preparing a configuration variables template for use with Microservices Container consists of the following general tasks. Note that whether you intend to use the template with a Microservices Container in a Docker container or on-premises may affect the order in which you complete these tasks:

Task	Description
1	Decide how you want to use the configuration variables template and how you want to supply values. Do you want to use the template with a Microservices Container running in a Docker container or an on-premises Microservices Container? Do you want to supply all the variable values in the template or do you want to use an environment variables as well? For more information, see "Approaches for Using a Configuration Variables Template with Microservices Container " on page 33.

Task	Description
2	Configure a Microservices Container such that the generated template will contain the properties that you want to control. When generating a template, the Microservices Container includes properties only for those assets which exist on the Microservices Container. That is, you cannot use a configuration variables template to add assets to the Microservices Container.
3	Generate the configuration variables template for the Microservices Container. For more information about creating a configuration variables template, see "Generating a Configuration Variables Template" on page 37.
4	Edit the configuration variables template to contain the property values that you want the Microservices Container to modify at startup. For more information about this stage, see "Editing a Configuration Variables Template" on page 39
5	Place the configuration variables template in the correct location with the appropriate name. The template location depends on whether or not you are using the template with a Docker image or on-premises Microservices Container and if the template is part of the image or file system. For more information, see "Approaches for Using a Configuration Variables Template with Microservices Container " on page 33. If you are using ENV variables as values in the configuration variables template, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers.
6	If you are using the template in a Docker container and the template resides inside the Microservices Container file system, build the Docker image. For more information about creating a Docker image, see the *webMethods Integration Server Administrator's Guide*

Generating a Configuration Variables Template

Microservices Container provides a utility to generate a configuration variables template based on the configuration of a running Microservices Container. This configuration variables template reflects the current configuration of the Microservices Container.

After the Microservices Container is configured, you can use the utility to generate the template. The resulting template serves as the starting point for creating a customized configuration variables template. Because a configuration variables template can be used to modify existing configuration on Microservices Container only, make sure you fully configure the Microservices Container for all facilities and features that you wish to be modified dynamically.

Before you generate the configuration variables template from a Microservices Container, you might want to:

- Make sure that the **Settings > Extended** page displays all of the server configuration parameters that you want to set with the template. To show settings, use Microservices Container Administrator to navigate to **Settings > Extended > Show and Hide Keys**. Select the check box next to each server configuration parameter that you want to appear in the template, and click **Save Changes**.
- Ensure that the keystores and truststores are loaded. When a keystore or truststore is loaded it indicates that the keystore or truststore configuration is valid and that the Microservices Container initialized the keystore or truststore alias successfully. To see if a keystore or truststore is loaded, navigate to the **Security > Keystore** page and ensure that the **Loaded** column displays **Yes**.

Note: You can create a configuration variables template manually. You do not need to generate a template from an existing configured Microservices Container. However, generating the template from a Microservices Container is a time-saving option that provides a starting point. A generated template may also be less prone to errors with key names in the template than a template created manually.

To generate a configuration variables template

1. Open the Microservices Container Administrator for the Microservices Container from which you want to generate the variables
2. In the **Microservices** menu of the Navigation panel, click **Configuration Variables**.
3. On the **Microservices > Configuration Variables** page, click **Generate Configuration Variables Template**.

   Microservices Container generates an application.properties file. The application.propeties file includes the text `#Sample Generated Template` followed by the date and time of the generation.
4. Download the application.properties file and save it to your preferred location for editing.

The generated application.proeperties file includes the following:

- The date and time that Microservices Container generated the template.
- Key-value pairs for asset properties for which a value is specified on the Microservices Container. The template does not include variables for assets that do not exist or for asset properties that are not specified.

For a list of assets and properties for which Microservices Container generates configuration variables see Configuration Variables Template Assets.

Note: In the template, any key that contain a period (.) as part of its name is escaped using another period. For example, for a JMS connection alias named `my.JMS.alias`, the template property corresponding to the Client ID in the alias the name: `jms.my..JMS..alias.clientID=value`

- Key-value pairs for the server configuration parameters that were configured to show on the Extended Settings page only.
- Keystore and truststore alias properties for keystores and truststores that were loaded at the time of template generation only.
- Encrypted values for any passwords that appear as property values in the template.
- For a configured asset, the configuration variables template lists properties only for which a value is specified. For example, if you specify a client ID for a JMS connection alias, but do not specify a user name or password for use with the JMS connection alias, the configuration variables template contains `jms.`*`aliasName`*`.clientID=`*`value`* but not `jms.`*`aliasName`*`.password` or `jms.`*`aliasName`*`.user`.

Editing a Configuration Variables Template

After you generate a configuration variables template from a Microservices Container, you modify the configuration variables template to contain the property values that you want applied to a Microservices Container at startup. You can hard code values or specify an environment variable as the value. You can also encrypt values to protect sensitive data.

Keep the following information in mind when you edit a configuration variables template.

- Any value that you specify must meet the requirements for the associated property. For example, global variable values cannot exceed 255 characters. If the value is not valid, failures will occur at runtime, either during server initialization or when clients start sending in requests.
- A configuration variables template can be used to change the configuration information for assets that already exist on the Microservices Container only. You cannot add assets to a Microservices Container by adding key-value pairs to a template.
- The configuration variables template lists only assets that existed on the Microservices Container from which you generated the template. You can add assets to the template if you know those assets will exist on the Microservices Container that will use the template. At startup, if the template specifies a value for a non-existent asset, the Microservices Container logs a message to the configuration variables log.

For more information about assets and properties and the corresponding key name, see Configuration Variables Template Assets.

- When Microservices Container generates an application.properties template, any properties that contain an alias name with a period (.) include another period as an escape character. For example, for a JMS connection alias named `my.JMS.alias`, the template property corresponding to the Client ID in the alias the name: `jms.my..JMS..Alias.clientID=value`

  Do not remove the period (.) acting as the escape character.

  If you edit the template by adding a key-value pair for an asset property and the asset property includes an alias name with a period (.), make sure to add another period to serve as the escape character.

To supply a value in the template

1. Open the template in a text editor.
2. For each value that you want to supply, do one of the following:

To...	Enter this for the property value...
Hard code a value	The value you want to use.
Use an environment value as the value	The name of the environment variable in the format: $env{*environmentVariableName* } Where *environmentVariableName* is the name of the environment variable. For example: `$env{JDBC_URL}`
Encrypt a value	Use Microservices Container Administrator to generate an encrypted value and then copy that value into the template. For more information see, "Encrypting Values for the Variables Template" on page 41. While Microservices Container accepts clear text passwords as well as encrypted ones, Software AG recommends that you encrypt all passwords and other sensitive data in your configuration variables templates.

3. Repeat the above step for each key that you want to set.
4. Save the template.

Encrypting Values for the Variables Template

You may want your configuration variables template to use encrypted values for sensitive data such as passwords. Microservices Container provides a way to generate an encrypted value which can then be placed in the template.

For encryption, Microservices Container uses password handles and the Password-Based Encryption technology installed with Microservices Container. A password handles associate a password on the host Microservices Container with a corresponding key (or handle). The handle is encrypted as an outbound password using the Password-Based Encryption (PBE) technology.

Note: When you generate a configuration variables template, Microservices Container encrypts any passwords that appear as property values in the template.

Software AG recommends that you encrypt all passwords and other sensitive data in your configuration variables templates.

To encrypt a value

1. Open the Microservices Container Administrator for the Microservices Container that you want to use to generate an encrypted value.
2. In the **Microservices** menu of the Navigation panel, click Configuration Variables.
3. On the **Microservices > Configuration Variables** page, click **Generate Encrypted Configuration Variables**.
4. On the **Microservices > Configuration Variables > Generate Encrypted Configuration Variable** page, in the **Value** field, enter the value that you want encrypted.

   By default, Microservices Container Administrator masks any characters that you enter in the **Value** field. Click the **Show Value** check box if you want Microservices Container Administrator to display the characters in clear text.
5. Click **Encrypt**.

   Microservices Container encrypts the value and displays the encrypted value in the **Encrypted Value** field.
6. Click **Copy** to copy the encrypted value to the clipboard.
7. Open the configuration variables template to which you want to add the encrypted value, locate the key for which you want to use the value, and then paste the copied value into the template.
8. Repeat steps 4 –7 for each value you want to encrypt.

Template File Locations

When the configuration variables template is ready for use with a Microservices Container, you need to save the template to a place accessible to Microservices Container upon startup. Where you place the configuration variables template file and any accompanying environment variables list depends on whether you are using the template with Microservices Container running in a Docker container or on-premises.

You can place the template file in one of the following locations.

- The location to be identified by the SAG_IC_CONFIG_PROPERTIES environment variable.
- *Integration Server_directory*/instances/*instanceName*

  For this option, the configuration variables template must be a file named application.properties. If you use this location with Microservices Container running in a Docker container, create the Docker image after completing the template and placing it here.
- *Integration Server_directory*

  For this option, the configuration variables template must be a file named application.properties. If you use this location with Microservices Container running in a Docker container, create the Docker image after completing the template and placing it here.

For more details about where to place the configuration variables template and an environment variables list, if you are using one, see "Approaches for Using a Configuration Variables Template with Microservices Container " on page 33.

Providing a Configuration Variables Template when Starting a Docker Container

When running a Microservices Container image in a Docker container, you can specify the configuration variables template and/or environment (ENV) variables in the `docker run` command.

Example

The following `docker run` command uses the `SAG_IS_CONFIG_PROPERTIES` ENV variable to specify the name and location of the configuration variables template. In this example, the Docker image for the Microservices Container is named is:microPkg and exposes ports 5555 and 9999. Additionally, the customApplication.properties file location is accessible by the Docker container.

```
docker run -d --name IS_Default -p 5555 -p 9999
-e SAG_IS_CONFIG_PROPERTIES=/opt/softwareag/customApplication.properties
is:microPkg
```

Example

The following `docker run` command uses the `SAG_IS_CONFIG_PROPERTIES` ENV variable to specify the name and location of the configuration variables template and uses the `--env-file` option to specify the list of ENV variables that are used in the configuration variables template. In this example, the Docker image for the Microservices Container is named is:microPkg and exposes ports 5555 and 9999.

```
docker run -d --name IS_Default -p 5555 -p 9999
-e SAG_IS_CONFIG_PROPERTIES=/opt/softwareag/customApplication.properties
--env-file env.list is:microPkg
```

Configuration Variables Logging

When Microservices Container applies the configuration variables template at startup, Microservices Container has not yet initialized the journal logger which is used for the server log. For this reason, Microservices Container uses a separate logging capability for logging Microservices Container messages related to processing a configuration variables template. The configuration variables log contains messages about the operations, warnings, and errors that occur while Microservices Container applies the template.

Below is an excerpt from the configuration variables log:

```
2018-06-18 15:34:10 EDT [ISS.0028.0016I] Configuration variables template file
found at C:\SoftwareAG\IntegrationServer\application.properties will be used for
processing.
2018-06-18 15:34:12 EDT [ISS.0028.0007I] Changed property in
C:\SoftwareAG\IntegrationServer\instances\default\config\server.cnf. Old value:
localhost. New value: localhost,127.0.0.1. Configuration variable key:
watt.net.proxySkipList.
2018-06-18 15:34:12 EDT [ISS.0028.0017W] Encryptor is creating password handle
because the given password handle
wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https does not exist in
the password store.
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0017W] Encryptor is creating password handle
because the given password handle
wm.is.admin.WSEndpoint.transport.CONSUMER.HTTPS.ws_cons_https does not exist in
the password store.
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.transport.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0007I] Changed property in
C:\SoftwareAG\IntegrationServer\instances\default\config\kerberos.cnf. Old
value: . New value:
```

Each log message is prefixed with the time stamp and a message ID that includes the logging facility (0028), message code, and the severity level. The severity levels are as follows:

	Level	Description	Example
C	Critical	Could not execute the configuration variables processing or an error occurred that caused Microservices Container initialization to stop.	Configuration file C:\SoftwareAG \IntegrationServer\instances \default\packages \WmPublic\config \listeners.cnf not found.
E	Error	One or more of the intended configuration changes could not be applied	java.io.FileNotFoundException: C:\SoftwareAG \IntegrationServer\instances \default\config\jndi \jndi_BasicAuth.properties (The system cannot find the file specified)
W	Warning	A configuration variable was applied, but there is something about which you should be aware.	Property corresponding to key username not found in XML file proxy.cnf.
I	Information	Informational messages about the progress of configuration variables processing.	Changed property in C:\SoftwareAG \IntegrationServer\instances \default\config\server.cnf. Old value: localhost. New value: localhost,127.0.0.1. Configuration variable key: watt.net.proxySkipList.
D	Debug	Low level messages, helpful for troubleshooting.	Processor found for the assetType truststore.

Microservices Container does not support filtering the contents of the configuration variables log based on logging level. By default, the configuration variables log includes all messages with a severity of Information or higher. To include debug messages in the log, you must set the `SAG_IS_CONFIG_VARIABLES_DEBUG` environment variable to true at startup.

Microservices Container writes the configuration variables log messages to the console (STDOUT) and/or to the location: *Integration Server_directory*/instances/*instanceName*/logs/configurationvariables.log. A Microservices Container running in a Docker container always writes the configuration variables log messages to both locations.

Note: Microservices Container running in a Docker container also writes the server log to the console. Configuration variables log messages include "ISS.0028" in the message ID which distinguishes the messages from server log messages.

When starting an on-premises Microservices Container from the command line, you can specify the destination for the configuration variables log using the `-log` switch.

Command Line Option	Log Destination
`-log none`	The console (STDOUT).
`-log both`	The console (STDOUT) and *Integration Server_directory*/instances/*instanceName* /logs/configurationvariables.log.

Note: If the `-log` switch is not specified, Microservices Container writes the log messages to this location only: *Integration Server_directory*/instances/*instanceName* /logs/configurationvariables.log

Starting Microservices Container from the command line is the same as starting Integration Server from the command line.

For information about starting Integration Server from the command line, see *webMethods Integration Server Administrator's Guide*.

Microservices Container does not preserve the contents of the configurations variable log across restarts. When running on-premises Microservices Container overwrites the existing configurations variable log upon startup.

Viewing the Applied Template for a Microservices Container

For a running Microservices Container you may want to see the configuration variables templates that Microservices Container applied at startup. This can help you compare current configuration to configuration at the time of startup. Using Microservices Container Administrator, you can obtain the application.properties file used at startup. If the application.properties used environment variables as values, Microservices Container substitutes the actual value for the environment variable value.

To view the applied configuration variables template

1. Open Microservices Container Administrator for which you want to view the applied template.
2. In the **Microservices** menu of the Navigation panel, click **Configuration Variables**.

3. On the **Microservices > Configuration Variables** page, click **Get Active Configuration Variables**.

   Microservices Container generates an application.properties file that contains the key-value pairs from the applied temple. The application.properties file includes the text `#No Active Template` if Microservices Container did not apply a template at startup.

6 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 Container 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 Container can establish a connection to the Consul server using the information provided in the alias.

To test a Consul server alias

1. Open Microservices Container 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 Container 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 Container 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 Container 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 Container Administrator prompts you to confirm deleting the alias. Click **OK**.

   If you delete the default Consul server alias, Microservices Container 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.

A Configuration Variables Template Assets

A configuration variables template is a properties file that contains configuration data as a series of key-value pairs where the key name reflects the asset and particular asset property for which you can supply a value. A configuration variables template contains key-value pairs for a subset of Microservices Container configuration only. That is, a configuration variables template does not include key-value pairs for all of the configuration information for a Microservices Container. Only some configuration information is supported for use with configuration variables templates.

Note: To use a configuration variables template in version 10.1, you must install IS_10.1_Core_Fix7 which includes PIE-52236, the feature that introduces support for configuration variables templates.

The following table identifies the assets and property names that correspond to the key names in a template.

Asset	Property	Key Name
Cache Manager	**Terracotta Server Array URLs**	`cachemanager.`*`cacheManagerName`*`.urls`
Email Port	**Host Name**	`email.`*`packageName`*`.`*`aliasName`*`.host`
	Password	`email.`*`packageName`*`.`*`aliasName`*`.password`
	Port	`email.`*`packageName`*`.`*`aliasName`*`.server_port`
	User Name	`email.`*`packageName`*`.`*`aliasName`*`.user`
File Polling Port	**Completion Directory**	`filepolling.`*`packageName`*`.`*`aliasName`*`.completionDir`
	Error Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`.errorDir`
	Monitoring Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`.monitorDir`
	Working Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`.workDir`

Asset	Property	Key Name
Global Variables	**Value**	`globalvariable.`*`globalVariableName`*`.value`
JDBC Pool Alias	**Database URL**	`jdbc.`*`aliasName`*`.dbURL`
	User ID	`jdbc.`*`aliasName`*`.userid`
	Password	`jdbc.`*`aliasName`*`.password`
JMS Connection Alias	**Connection Client ID**	`jms.`*`aliasName`*`.clientID`
	User	`jms.`*`aliasName`*`.user`
	Password	`jms.`*`aliasName`*`.password`
JNDI Provider Alias	**Provider URL**	`jndi.`*`aliasName`*`.providerURL`
	Provider URL Failover List	`jndi.`*`aliasName`*`.providerURLFailoverList`
	Security Principal	`jndi.`*`aliasName`*`.securityPrincipal`
	Security Credentials	`jndi.`*`aliasName`*`.securityCredentials`
Kerberos	**Realm**	`kerberos.KerberosConfig.kerberosConfig`
	Key Distribution Center Host	`kerberos.KerberosConfig.kerberosKDC`
	Kerberos Configuration File	`kerberos.KerberosConfig.kerberosRealm`
	Use Subject Credentials Only	`kerberos.KerberosConfig.useSubjectCreds`

Asset	Property	Key Name
Keystore Alias	**Location**	`keystore.`*`aliasName`*`.ksLocation`
	Password	`keystore.`*`aliasName`*`.ksPassword`
	Key Alias Password	`keystore.`*`aliasName`*`.keyAlias.`*`keyAliasName`*`.keyAliasPassword`
LDAP Configuration	**Directory URL**	`ldap.`*`ldapDirectory`*`.url`
	Principal	`ldap.`*`ldapDirectory`*`.prin`
	Credentials	`ldap.`*`ldapDirectory`*`.password`
Proxy Server Alias	**Host Name or IP Address**	`proxyserver.`*`aliasName`*`.host`
	Port Number	`proxyserver.`*`aliasName`*`.port`
	User Name	`proxyserver.`*`aliasName`*`.username`
	Password	`proxyserver.`*`aliasName`*`.password`
Proxy Server Bypass	**Addresses**	`settings.watt.net.proxySkipList`
Remote Server Alias	**Host Name or IP Address**	`remoteserver.`*`aliasName`*`.host`
	Port Number	`remoteserver.`*`aliasName`*`.port`
	User Name	`remoteserver.`*`aliasName`*`.user`
	Password	`remoteserver.`*`aliasName`*`.password`
	Retry Server	`remoteserver.`*`aliasName`*`.retryServer`
Server configuration	Any	`settings.`*`serverConfigurationParameterName`

Asset	Property	Key Name
parameters (server.cnf)		
SFTP Server Alias	**Host Name or IP Address**	`sftpserver.`*`aliasName`*`.hostName`
	Port Number	`sftpserver.`*`aliasName`*`.port`
	Proxy Alias	`sftpserver.`*`aliasName`*`.proxyAlias`
SFTP User Alias	**User Name**	`sftpuser.`*`aliasName`*`.userName`
	Password	`sftpuser.`*`aliasName`*`.password`
	SFTP Server Alias	`sftpuser.`*`aliasName`*`.sftpServerAlias`
Truststore Alias	**Location**	`truststore.`*`aliasName`*`.ksLocation`
	Password	`truststore.`*`aliasName`*`.ksPassword`
webMethods Cloud Account	**Stage**	`wmcloudaccount.`*`webMethodsCloudAccountAlias`*`.stage`
	Allowed On-Premise Hosts	`wmcloudaccount.`*`webMethodsCloudAccountAlias`*`.onPremiseHosts`
webMethods Cloud Settings	**User Name**	`wmcloudsettings.default.username`
	Password	`wmcloudsettings.default.password`
	webMethods Cloud URL	`wmcloudsettings.default.iLiveURL`
Web Service Endpoint Alias	**Host Name or IP Address**(HTTP/	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`*`aliasName`*`.transportInfo.host`

Asset	Property	Key Name
	S Transport Properties)	
	Port Number (HTTP/S Transport Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.` *`aliasName`*`.transportInfo.port`
	User Name (HTTPS Transport Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.` *`aliasName`*`.transportInfo.user`
	Password (HTTPS Transport Properties)	`webserviceendpoint.consumer.`*`protocol`*`.` *`aliasName`*`.transportInfo.transportPassword`
	User Name (WS Security Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.` *`aliasName`*`.messageInfo.user`
	Password (WS Security Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.` *`aliasName`*`.messageInfo.messagePassword`

	To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`
		`aliasName``.messageaddressingproperties.`
		`toMsgAddr`

	From (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`
		`aliasName``.messageaddressingproperties.`
		`fromMsgAddr`

	Reply To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`
		`aliasName``.messageaddressingproperties.`
		`replyToMsgAddr`

	Fault To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`*`aliasName`*`.` `messageaddressingproperties.faultToMsgAddr`

Asset	Property	Key Name
	Where *type* can be consumer, provider, or messageaddressing and *protocol* can be HTTP, HTTPS, or JMS.	