

Developing Microservices with webMethods Microservices Runtime

Version 10.3

October 2018

This document applies to webMethods Microservices Runtime Version 10.3 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.

Copyright © 2017-2018 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Table of Contents

About this Guide.....	5
Document Conventions.....	5
Online Information and Support.....	6
Data Protection.....	7
Getting Started with webMethods Microservices Runtime.....	9
What Are Microservices?.....	10
What Is webMethods Microservices Runtime?.....	10
What Is Microservices Runtime Administrator?.....	12
Starting, Shutting Down, and Restarting Microservices Runtime.....	13
Starting Microservices Runtime and Microservices Runtime Administrator.....	14
Starting Microservices Runtime on Windows.....	14
Starting Microservices Runtime on UNIX.....	14
Starting Microservices Runtime Administrator.....	14
Shutting Down Microservices Runtime.....	15
Shutting Down Microservices Runtime on Windows.....	15
Shutting Down Microservices Runtime from Microservices Runtime Administrator on Windows or UNIX.....	15
Restarting Microservices Runtime.....	16
Configuring Microservices Runtime.....	17
Using a Circuit Breaker with Services.....	19
About Circuit Breaker.....	20
Circuit States.....	20
How Does a Circuit Breaker for a Service Work?.....	21
Configuring a Circuit Breaker for a Service.....	24
Building a Service for Use with an Open Circuit.....	25
Configuring the Circuit Breaker Thread Pool.....	26
Circuit Breaker Statistics.....	28
Automatic Package Deployment.....	31
How Automatic Package Deployment Works.....	32
Determining Package Dependencies During Automatic Deployment.....	33
Considerations for Auto Deployment of Packages.....	33
Enabling and Configuring Automatic Package Deployment.....	34
Automatic Package Deployment Location.....	35
Using Configuration Variables Templates with Microservices Runtime.....	37
About Configuration Variables Templates.....	38
What Does a Configuration Variables Template Look Like?.....	38
When Is the Template Applied?.....	40

Approaches for Using a Configuration Variables Template with Microservices Runtime.....	41
Overview of Building a Configuration Variables Template.....	44
Generating a Configuration Variables Template.....	45
Editing a Configuration Variables Template.....	47
Encrypting Values for the Variables Template.....	48
Template File Locations.....	49
Providing a Configuration Variables Template when Starting a Docker Container.....	50
Configuration Variables Logging.....	51
Viewing the Applied Template for a Microservices Runtime.....	53
Monitoring Microservices Runtime.....	55
Overview of Monitoring Microservices Runtime.....	56
About the Health Gauge.....	56
Predefined Health Indicators.....	57
Enabling and Disabling Health Indicators.....	58
Health Indicator Properties.....	58
Configuring Health Indicator Properties.....	60
Invoking the Health Gauge.....	60
Obtaining Metrics for a Microservices Runtime.....	61
Invoking the Metrics Endpoint.....	61
Consul Support.....	63
Configuring Connections to Consul Server.....	64
Testing an Alias for the Consul Server.....	65
Setting the Default Alias for the Consul Server.....	66
Deleting a Consul Server Alias.....	66
Consul Public Services Folder.....	66
pub.consul.client:deregisterService.....	67
pub.consul.client:getAllHostsForService.....	68
pub.consul.client:getAnyHostForService.....	68
pub.consul.client:registerService.....	69
Configuration Variables Template Assets.....	71
Prometheus Metrics.....	77
Prometheus Metrics Returned by Microservices Runtime.....	78
Server Metrics.....	78
Service Metrics.....	80

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 service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	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.

Convention	Description
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

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

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to "empower@softwareag.com" with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at "<https://empower.softwareag.com/>".

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](#)".

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at "https://empower.softwareag.com/public_directory.asp" and give us a call.

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.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Getting Started with webMethods Microservices Runtime

■ What Are Microservices?	10
■ What Is webMethods Microservices Runtime?	10
■ What Is Microservices Runtime Administrator?	12

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?

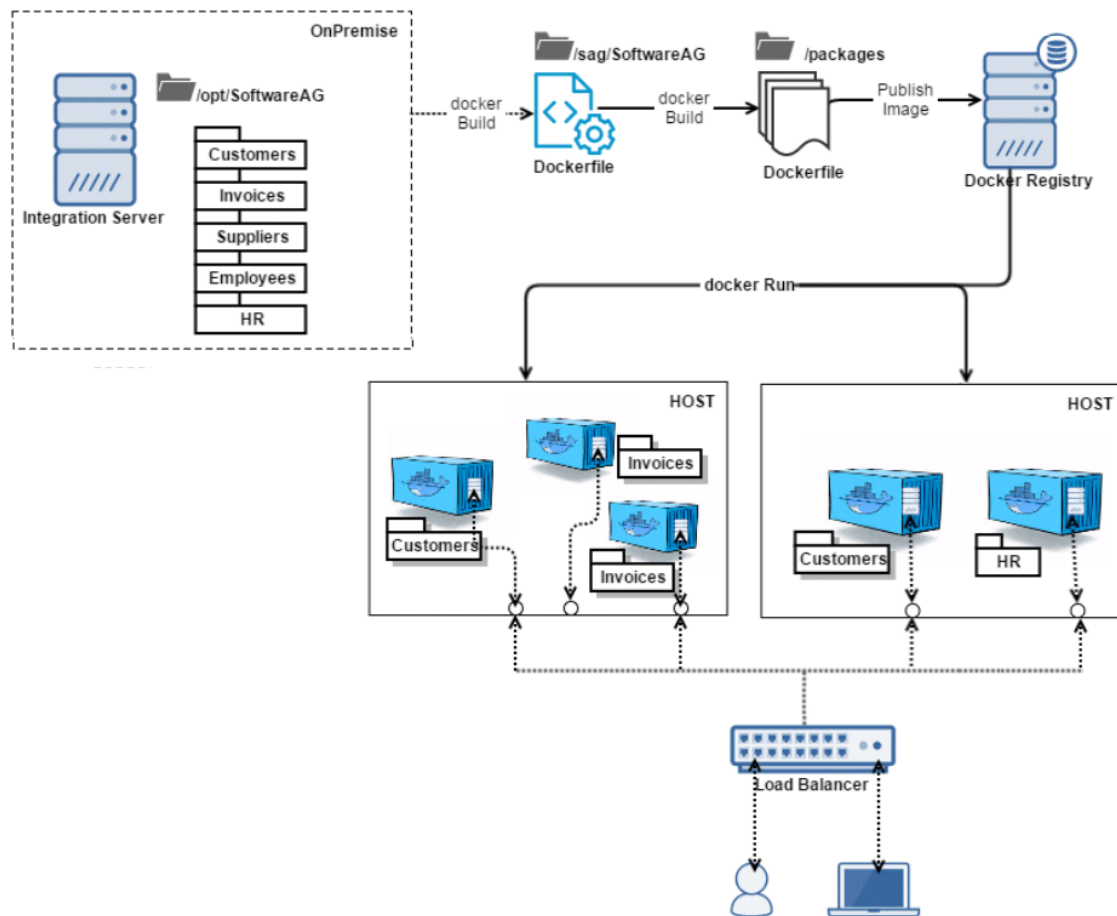
Software AG offers a lightweight container called webMethods Microservices Runtime to host microservices you develop in Software AG Designer. Using Microservices Runtime, you can deliver microservices as an Integration Server package that includes a set of related services, interfaces, document types, 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	14
■ Shutting Down Microservices Runtime	15
■ Restarting Microservices Runtime	16

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 Microservices Runtime > StartMicroservices RuntimeinstanceName**.

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 > Microservices Runtime Administrator > Microservices Runtime Administrator for *instanceName***.
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 Microservices Runtime > Stop Microservices Runtime *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 <i>number</i> 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 <i>number</i> 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.

The following table identifies additional components that you might have installed with Microservices Runtime and provides the name of the guide that contains more information about configuring the component.

Component	See this guide for more information
CentraSite Asset Publisher Support	<i>webMethods Service Development Help</i>
Common Directory Service Support	<i>webMethods Integration Server Administrator's Guide</i>
Digital Event Services Support	<i>webMethods Integration Server Administrator's Guide</i>
Event Routing Support	<i>Communicating Between Software AG Products Using Event Routing</i>
External RDBMS Support	<i>webMethods Integration Server Administrator's Guide</i>

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. For instructions on creating and running Docker images for Microservices Runtime, see *webMethods Integration Server Administrator's Guide*.

4 Using a Circuit Breaker with Services

■ About Circuit Breaker	20
■ How Does a Circuit Breaker for a Service Work?	21
■ Configuring a Circuit Breaker for a Service	24
■ Building a Service for Use with an Open Circuit	25
■ Configuring the Circuit Breaker Thread Pool	26
■ Circuit Breaker Statistics	28

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 microservices 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

The state of a circuit for a service determines how circuit breaker responds to a request to invoke the service.

The following table identifies how the circuit breaker responds to an invocation request based on the circuit state.

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	<p>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.</p> <p>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.</p>

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	<p>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.</p> <ul style="list-style-type: none"> ■ 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	<p>The circuit breaker examines the state of the circuit for the service.</p> <ul style="list-style-type: none"><li data-bbox="492 415 1341 579">■ 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.<li data-bbox="492 604 1341 737">■ 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.<li data-bbox="492 762 1341 856">■ 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. <p>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.</p>
3	<p>When the circuit state is closed, upon execution of the service by the circuit breaker, one of the following occurs:</p> <ul style="list-style-type: none"><li data-bbox="492 1119 971 1150">■ The service executes successfully.<div data-bbox="540 1167 1365 1331"><p>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.</p></div><li data-bbox="492 1356 1341 1541">■ The service ends with an exception.<p>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.</p><li data-bbox="492 1566 1357 1730">■ 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.<p>If the Cancel thread on timeout property is set to true, the circuit breaker attempts to cancel the thread executing the service.</p><p>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</p>

Step	Description
	<p>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.</p>
4	<p>Subsequent invocations of the service result in the circuit breaker opening or “tripping” the circuit for the service if:</p> <ul style="list-style-type: none"> ■ The number of failure events for the service equals the Failure threshold property value <i>and</i> ■ 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. <p>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.</p> <p>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.</p>
5	<p>When the circuit breaker receives a request to execute the service, the circuit breaker does one of the following:</p> <ul style="list-style-type: none"> ■ 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 <code>\$circuitBreakerService</code> and <code>\$circuitBreakerEvent</code> parameters in the input pipeline for the alternate service. For more information about the circuit open

Step	Description
	<p>service, see the section <i>Building a Service for Use with an Open Circuit</i>.</p> <p>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.</p>
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	<p>The circuit breaker receives a request for the service.</p> <ul style="list-style-type: none"> ■ 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: <ul style="list-style-type: none"> ■ 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 parameters and values in the service pipeline for use by the open circuit service.

The following identifies and describes the parameters that circuit breaker places in the pipeline for an open circuit service.

Parameter Name	Description
<i>\$circuitBreakerService</i>	String. Fully qualified name of the service with the open circuit.
<i>\$circuitBreakerEvent</i>	<p>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 <i>\$circuitBreakerEvent</i> parameter has one of the following values:</p> <ul style="list-style-type: none"> ■ <code>CIRCUIT_OPENED</code> if this is the first execution of this open circuit service since the circuit opened. ■ <code>CIRCUIT_OPEN</code> 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 parameters in the above table 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 cancel 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.

The following table identifies the circuit breaker information that Microservices Runtime maintains for each service with a configured circuit breaker.

Field	Description
Name	Name of the service for which a circuit breaker is configured.
State	<p>The state of the circuit. The circuit state can be one of the following:</p> <ul style="list-style-type: none"> ■ 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. <p>For a detailed explanation of possible circuit states, see the “Circuit States” on page 20Circuit States.</p>
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

Field	Description
	circuit, see “How Does a Circuit Breaker for a Service Work?” on page 21 <i>How Does a Circuit Breaker for a Service Work?</i> .
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 Automatic Package Deployment

■ How Automatic Package Deployment Works	32
■ Determining Package Dependencies During Automatic Deployment	33
■ Considerations for Auto Deployment of Packages	33
■ Enabling and Configuring Automatic Package Deployment	34

Automatic package deployment allows Microservices Runtime to install or upgrade packages automatically. When automatic package deployment is enabled, packages can be installed or upgraded without using Deployer or an administrator tool such as, Microservices Runtime Administrator. Automatic deployment can be used with an on-premises Microservices Runtime and an Microservices Runtime that runs inside a Docker container.

Automatically deploying packages can be particularly useful for a Microservices Runtime that runs in a Docker container. Instead of creating the Docker image containing individual Packages, creating a base image containing Default package can be used to bring in custom packages. Packages to be installed or updated on the container can be placed in an `autodeploy` directory that Microservices Runtime periodically scans. Microservices Runtime will find and install the new or updated packages. If you do not use automatic deployment, you must rebuild the Docker image to use any new or updated packages with Microservices Runtime container.

Note: The automatic package deployment feature is available by default for a webMethods Microservices Runtime. To use the automatic package deployment feature with Integration Server, your Integration Server must have additional licensing.

How Automatic Package Deployment Works

In automatic package deployment, the packages that you want Microservices Runtime to deploy automatically are placed in a location that Microservices Runtime periodically scans for new or updated custom packages. Microservices Runtime executes a system task named “Auto Package Deployer” that scans the folder and checks for new or updated custom packages. When Microservices Runtime finds new or updated custom packages, Microservices Runtime does one of the following:

- For a new package, Microservices Runtime installs, archives, and activates the package.
- For an updated custom package, Microservices Runtime does one of the following:
 - If hot deployment is enabled for automatic package deployment, Microservices Runtime uses the block-and-wait approach employed by hot deployment to deploy the package. This ensures that Microservices Runtime assets are available for processing without any noticeable downtime. For more information about how hot deployment works, see *webMethods Integration Server Administrator's Guide*
 - If hot deployment is not enabled, Microservices Runtime unloads the existing package information from memory and loads the new version of the package and its contents into memory. While the package is being upgraded, it is possible that some assets might be unavailable for some time. This could cause serious issues such as denial of service or disruption of the in-flight tasks.

Hot deployment is enabled for automatic package deployment if either of the following is true:

- Hot deployment is enabled globally for the Microservices Runtime. Hot deployment is enabled globally when the **Enabled** field is set to **Yes** on the **Settings > Hot Deployment** page.
- Hot deployment is enabled for automatic package deployment when the `watt.server.autodeploy.alwaysUseHotDeployment` server configuration parameter is set to `true`.

By default, the automatic package deployment feature is disabled in Microservices Runtime. To enable automatic package deployment or configure other aspects of hot deployment, such as the system task execution frequency, see the section *Enabling and Configuring Automatic Package Deployment*.

Note: To identify packages for automatic deployment, you must manually copy the packages to the `Integration Server_directory\instances\instance_name\default\replicate\autodeploy` folder.

Determining Package Dependencies During Automatic Deployment

If you are deploying new package or updated custom packages and it has new dependent packages, you have to deploy the dependent packages too in the `autodeploy` folder. If you are deploying a new package or updated custom packages with existing dependent packages that are already deployed, then Microservices Runtime activates the new package. If you are deploying a new or updated custom packages with missing package dependencies the activation will fail. Also, if you are using hot deployment then you should identify package dependencies correctly. For more information see *webMethods Integration Server Administrator's Guide*.

Considerations for Auto Deployment of Packages

Before you configure automatic package deployment in Microservices Runtime, keep the following behavior limitations and considerations in mind.

- You cannot cancel an in-progress automatic package deployment operation.
- If the `watt.server.autodeploy.alwaysUseHotDeployment` parameter is set to `true`. Microservices Runtime performs hot deployment for the installation of any custom package regardless of whether it is a new package or an upgraded package. For more information on hot deployment, see [“Enabling and Configuring Automatic Package Deployment” on page 34](#).
- If you are deploying new package and it has new dependent packages, you have to deploy all at once in the **autodeploy** folder.

- If you use hot deployment for automatic package deployment, all of the hot deployment considerations apply to automatic package deployment as well. For more information about hot deployment considerations, see [“Enabling and Configuring Automatic Package Deployment”](#) on page 34.
- When hot deployment is used for automatic package deployment, Microservices Runtime uses the configured hot deployment values. For more information about configuring hot deployment, see [“Enabling and Configuring Automatic Package Deployment”](#) on page 34.

Enabling and Configuring Automatic Package Deployment

Microservices Runtime uses various configuration settings for automatic package deployment of new and updated packages. Most of the settings have defaults. You can change these setting by using the **Settings > Extended** screen of the Integration Server Administrator as described below.

To configure automatic deployment for Microservices Runtime

1. In the Microservices Runtime Administrator, under **Settings**, click **Extended**.
2. Click **Edit Extended Settings**.
3. If the `watt.server.autodeploy.alwaysUseHotDeployment`, `watt.server.autodeploy.enabled`, and `watt.server.autodeploy.interval` parameters do not appear the **Extended Settings** list, do the following:
 - a. Select **Show and Hide Keys**.
 - b. Select the check boxes next to the `watt.server.autodeploy.alwaysUseHotDeployment`, `watt.server.autodeploy.enabled`, and `watt.server.autodeploy.interval` parameters.
 - c. Click **Save Changes**.
4. On the **Settings > Extended** page, click **Edit Extended Settings**.
5. Set the following server configuration properties in Microservices Runtime.

For this extended setting	Description
<code>watt.server.autodeploy.enabled=true</code>	Enables automatic package deployment .
<code>watt.server.autodeploy.alwaysUseHotDeployment=true</code>	Use hot deployment for automatic deployment of packages.

For this extended setting	Description
watt.server.autodeploy.interval=5	Specifies the interval, measured in minutes, at which Microservices Runtime executes the autodeploy system task.

6. Click **Save Changes**.
7. Restart the Microservices Runtime.

The updated settings are now in effect.

Automatic Package Deployment Location

Where you place custom packages for automatic package deployment depends on whether you are using an on-premises Microservices Runtime or an Microservices Runtime running in a Docker container.

- For an on-premises Microservices Runtime, place packages that you want automatically deployed in this location:
Integration Server_directory\instances\instance_name \replicate\autodeploy folder.
- For a Microservices Runtime running a Docker container, when you start the Docker container, you can map the above folder of a container to a volume which points to a folder on your HOST machine. You can then place packages that you want to the server to automatically deploy in the folder specified by the volume.

6 Using Configuration Variables Templates with Microservices Runtime

■ About Configuration Variables Templates	38
■ What Does a Configuration Variables Template Look Like?	38
■ When Is the Template Applied?	40
■ Approaches for Using a Configuration Variables Template with Microservices Runtime	41
■ Overview of Building a Configuration Variables Template	44
■ Generating a Configuration Variables Template	45
■ Editing a Configuration Variables Template	47
■ Template File Locations	49
■ Providing a Configuration Variables Template when Starting a Docker Container	50
■ Configuration Variables Logging	51
■ Viewing the Applied Template for a Microservices Runtime	53

About Configuration Variables Templates

Microservices Runtime provides the ability to create a Docker image from an installed and configured instance of Microservices Runtime. The Docker image contains the Microservices Runtime application including packages and Microservices Runtime 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 Runtime 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 Runtime stores configuration information inside its file system and therefore inside any Docker image created from the Microservices Runtime, some configuration information can be externalized and passed to the Microservices Runtime at startup. With Microservices Runtime 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 Runtime. The property values can be set externally in the template and then passed to a Microservices Runtime when it starts up. As part of the startup process, Microservices Runtime 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 Runtime 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 Runtime 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 Runtime. To use the configuration variables feature with Integration Server, your Integration Server must have additional licensing.

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 Runtime, the template is populated with configuration data from the Microservices Runtime instance. The template content reflects the Microservices Runtime configuration at the time Microservices Runtime 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 Runtime. 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/Y1lw1XqEK0B17RFucP0kgMHNt823RUU6ad39tw\=
jdbc.webMPool.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 Runtime Administrator provides a utility to encrypt a value.

Microservices Runtime Administrator uses password handles and the Password-Based Encryption technology installed with Microservices Runtime 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 Runtime applies a configuration variables template at startup. When Microservices Runtime starts, either in a Docker container or on-premises, Microservices Runtime 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 Runtime locates a template, Microservices Runtime does not look for a template in any remaining locations.

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

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

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

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

Approaches for Using a Configuration Variables Template with Microservices Runtime

Because a configuration variables template can be used with a Microservices Runtime running in a Docker container or an on-premises Microservices Runtime, 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 Runtime so that Microservices Runtime applies the template.

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

- Will the Microservices Runtime with which you want to use the template run in a Docker container or on-premises (outside of a Docker container)?
- If Microservices Runtime 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 Runtime so that Microservices Runtime applies the template values.

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

Approach	Description and requirements
Docker image does not include template which does not use ENV variables	Microservices Runtime 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

Approach	Description and requirements
	<p>Template location: A location that is accessible to the Docker container.</p> <p>Start up: The <code>docker run</code> command includes the <code>SAG_IS_CONFIG_PROPERTIES</code> environment variable which specifies the name and location of the configuration variables template.</p>
<p>Docker image does not include template and template uses ENV variables</p>	<p>Microservices Runtime 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.</p> <p>Template file name: <code>application.properties</code> or a user-defined name</p> <p>Template location: A location that is accessible to the Docker container.</p> <p>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.</p> <p>Start up: The <code>docker run</code> command includes the <code>SAG_IS_CONFIG_PROPERTIES</code> environment variable which specifies the name and location of the configuration variables template. If using the <code>docker run</code> command, the command must also include options <code>--env</code>, <code>-e</code> or <code>--env-file</code> options for setting the ENV variables in the container.</p>
<p>Docker image includes template which uses ENV variables</p>	<p>Microservices Runtime 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.</p> <p>Template name: <code>application.properties</code></p> <p>Template location is one of the following:</p> <ul style="list-style-type: none"> ■ <code>Integration Server_directory/instances/instanceName/application.properties</code> ■ <code>Integration Server_directory/application.properties</code> <p>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</p>

Approach	Description and requirements
	<p>more information about how to specify the ENV variables for Docker containers.</p> <p>Start up: The <code>docker run</code> command includes the <code>SAG_IS_CONFIG_PROPERTIES</code> environment variable which specifies the name and location of the configuration variables template. The command must also include options <code>--env</code>, <code>-e</code> or <code>--env-file</code> options for setting the ENV variables in the container.</p>
<p>On-premises Microservices Runtime installation does not include template and template does not use ENV variables.</p>	<p>Microservices Runtime runs on-premises, configuration variables values are hard coded in the template only, and the template is not included in the Microservices Runtime file system. The template does not use ENV variables.</p> <p>Template name: <code>application.properties</code> or a user-defined name</p> <p>Template location: A location that is accessible to the Microservices Runtime.</p> <p>Start up: Microservices Runtime start up includes the <code>SAG_IS_CONFIG_PROPERTIES</code> ENV variable which specifies the name and location of the configuration variables template.</p>
<p>On-premises Microservices Runtime includes template which does not use ENV variables.</p>	<p>Microservices Runtime runs on-premises and a configuration variables template defines all the values. The template does not use ENV variables.</p> <p>Template name: <code>application.properties</code></p> <p>Template location is one of the following:</p> <ul style="list-style-type: none"> ■ <code>Integration Server_directory/instances/instanceName/application.properties</code> ■ <code>Integration Server_directory/application.properties</code> <p>Start up: This approach does not require any special steps at start up.</p>
<p>On-premises Microservices Runtime includes template which uses ENV variables</p>	<p>Microservices Runtime runs on-premises and a configuration variables template hard codes the values and/or uses environment variables for the values.</p> <p>Template name: <code>application.properties</code></p> <p>Template location is one of the following:</p>

Approach	Description and requirements
	<ul style="list-style-type: none"> ■ <i>Integration Server_directory?/instances/instanceName / application.properties</i> ■ <i>Integration Server_directory/application.properties</i> <p>Environment variables can be specified in <i>Integration Server_directory/profiles/IS_instanceName /bin/custom_setenv.bat/sh</i> files or ENV variables can be pre-defined in your environment.</p> <p>Start up: This approach does not require any special steps at start up.</p>

Overview of Building a Configuration Variables Template

Preparing a configuration variables template for use with Microservices Runtime consists of a series of general tasks. The order in which you complete these tasks may be affected by whether you intend to use the template with a Microservices Runtime in a Docker container or an on-premises Microservices Runtime.

The following table identifies the general tasks that you complete for preparing a configuration variables template for use with Microservices Runtime.

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 Runtime running in a Docker container or an on-premises Microservices Runtime? 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 Runtime” on page 41.
2	Configure a Microservices Runtime such that the generated template will contain the properties that you want to control. When generating a template, the Microservices Runtime includes properties only for those assets which exist on the Microservices Runtime. That is, you cannot use a configuration variables template to add assets to the Microservices Runtime.
3	Generate the configuration variables template for the Microservices Runtime. For more information about creating a configuration variables template, see “Generating a Configuration Variables Template” on page 45.

Task	Description
4	<p>Edit the configuration variables template to contain the property values that you want the Microservices Runtime to modify at startup. For more information about this stage, see “Editing a Configuration Variables Template” on page 47</p>
5	<p>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 Runtime 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 Runtime ” on page 41.</p> <p>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.</p>
6	<p>If you are using the template in a Docker container and the template resides inside the Microservices Runtime file system, build the Docker image.</p> <p>For more information about creating a Docker image, see <i>webMethods Integration Server Administrator’s Guide</i>.</p> <p>For more information about creating a Docker image, see the <i>webMethods Integration Server Administrator’s Guide</i></p>

Generating a Configuration Variables Template

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

After the Microservices Runtime 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 Runtime only, make sure you fully configure the Microservices Runtime for all facilities and features that you wish to be modified dynamically.

Before you generate the configuration variables template from a Microservices Runtime, 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 Runtime 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 Runtime 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 Runtime. However, generating the template from a Microservices Runtime 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 Runtime Administrator for the Microservices Runtime 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 Runtime generates an application.properties file. The application.properties 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.properties file includes the following:

- The date and time that Microservices Runtime generated the template.
- Key-value pairs for asset properties for which a value is specified on the Microservices Runtime. 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 Runtime 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 Runtime, you modify the configuration variables template to contain the property values that you want applied to a Microservices Runtime 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 Runtime only. You cannot add assets to a Microservices Runtime by adding key-value pairs to a template.
- The configuration variables template lists only assets that existed on the Microservices Runtime from which you generated the template. You can add assets to the template if you know those assets will exist on the Microservices Runtime that will use the template. At startup, if the template specifies a value for a non-existent asset, the Microservices Runtime 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 Runtime 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	<p>The name of the environment variable in the format: <code>\$env{environmentVariableName}</code></p> <p>Where <i>environmentVariableName</i> is the name of the environment variable. For example:</p> <pre>\$env{JDBC_URL}</pre>
Encrypt a value	<p>Use Microservices Runtime 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 48.</p> <p>While Microservices Runtime 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.</p>

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 Runtime provides a way to generate an encrypted value which can then be placed in the template.

For encryption, Microservices Runtime uses password handles and the Password-Based Encryption technology installed with Microservices Runtime. A password handle associates a password on the host Microservices Runtime 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 Runtime 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 Runtime Administrator for the Microservices Runtime 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 Runtime Administrator masks any characters that you enter in the **Value** field. Click the **Show Value** check box if you want Microservices Runtime Administrator to display the characters in clear text.

5. Click **Encrypt**.

Microservices Runtime 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 Runtime, you need to save the template to a place accessible to Microservices Runtime 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 Runtime 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 Runtime 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 Runtime 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 Runtime”](#) on page 41.

Providing a Configuration Variables Template when Starting a Docker Container

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

Level	Description	Example
C	Critical	Could not execute the configuration variables processing or an error occurred that caused Microservices Runtime initialization to stop.
E	Error	One or more of the intended configuration changes could not be applied

Level	Description	Example
		\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.
I	Information	Informational messages about the progress of configuration variables processing.
D	Debug	Low level messages, helpful for troubleshooting.

Microservices Runtime 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 Runtime 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 Runtime running in a Docker container always writes the configuration variables log messages to both locations.

Note: Microservices Runtime 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 Runtime from the command line, you can specify the destination for the configuration variables log using the `-log` switch.

The following table identifies the command line options for specifying the destination of the configuration variables log file.

Command Line Option	Log Destination
<code>-log none</code>	The console (STDOUT).

Command Line Option	Log Destination
<code>-log both</code>	The console (STDOUT) and <i>Integration Server_directory/instances/instanceName /logs/configurationvariables.log</i> .

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

Starting Microservices Runtime 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*.

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

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

Viewing the Applied Template for a Microservices Runtime

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

To view the applied configuration variables template

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

7 Monitoring Microservices Runtime

■ Overview of Monitoring Microservices Runtime	56
■ About the Health Gauge	56
■ Obtaining Metrics for a Microservices Runtime	61

Overview of Monitoring Microservices Runtime

Microservices Runtime provides capabilities for monitoring the health of a Microservices Runtime and gathering metrics about the server and the microservices it contains. External applications such as container management and monitoring tools, can use the metrics and data supplied by Microservices Runtime to help determine whether the Microservices Runtime is performing correctly or optimally.

Microservices Runtime exposes the monitoring features via endpoints. Requests sent to the endpoint result in the invocation of an internal service that gathers data and then returns the status and/or payload do the requester.

Microservices Runtime includes the following monitoring capabilities:

- Health gauge which returns an overall up or down status for a Microservices Runtime based on a set of health indicators. The health gauge endpoint is `http://host:port/health`
- Metrics which returns server and service metrics in Prometheus configuration format. The metrics endpoint is `http://host:port/metrics`

Note: The monitoring features are available by default for Microservices Runtime. To use the monitoring features with Integration Server, your Integration Server must have additional licensing.

About the Health Gauge

The health gauge returns an overall UP or DOWN status for the Microservices Runtime based on the collective status of enabled health indicators. When the `health` endpoint is invoked, Microservices Runtime executes all of the enabled health indicators. A health indicator determines the UP or DOWN status of a specific component of Microservices Runtime. If all of the health indicators return an UP status, the entire Microservices Runtime is considered to be up. The health gauge returns a HTTP 200 status code to the requester. If even one of the health indicators returns a DOWN status, the entire Microservices Runtime is considered to be down. The health gauge returns a HTTP 503 status code to the requester.

Regardless of the overall status, the response includes a payload in JSON format that contains more details for each health indicator, including whether the indicator returned a status of UP or DOWN. For example, the `ServiceThread` health indicator returns the current number of service threads in use, the current number of available threads, and the maximum number of available threads.

Predefined Health Indicators

Microservices Runtime includes predefined health indicators for some of the basic components of a Microservices Runtime. Some, but not all, of the health indicators have a configurable property that you can use to specify the threshold at which a health indicator returns an UP or DOWN status.

The following table describes the predefined health indicators included with Microservices Runtime.

Indicator Name	Description
Diskspace	Checks for low disk space. Returns status of UP if the percentage of free disk space is greater than what is specified in the Free disk space threshold property for the Diskspace health indicator. Otherwise, returns a status of DOWN.
JDBC	Checks for available JDBC connections across all JDBC functions such as ISInternal and ISCoreAudit. Returns a status of UP if, for each JDBC connection pool, Microservices Runtime can obtain a valid JDBC connection before a 200 millisecond time out elapses. Otherwise, returns a status of DOWN. The timeout value is not configurable. The JDBC health indicator skips any JDBC functions that do not have an associated pool.
JMS	Checks that JMS connection alias are available. Returns a status of UP if all enabled JMS connection aliases are active, meaning that Microservices Runtime can ping the JMS Provider or create a connection successfully. Otherwise, returns a status of DOWN.
JNDIAliases	Checks that the connections for a JNDI aliases are up by attempting to make a connection for JNDI.
Memory	Checks for low available memory. Returns a status of UP if the percentage of free memory is greater than what is specified for the Free memory threshold property for the Memory health indicator. Otherwise, returns a status of DOWN.
RemoteServers	Checks the status of remote servers. Returns a status of UP if Microservices Runtime can successfully invoke the internal service <code>wm.server:ping</code> on each server for which there exists a remote server alias. Otherwise, returns a status of DOWN.
SFTPServers	Checks the connection to remote SFTP serves for which an SFTP server alias is configured. Returns a status of UP if a connection

Indicator Name	Description
	can be obtained for all of SFTP server aliases with at least one SFTP user alias.
ServiceThread	Checks for low available server threads. Returns a status of UP if the percentage of available server threads is greater than what is specified in the Available threads threshold property for the ServiceThread health indicator. Otherwise, returns a status of DOWN.
Sessions	Checks for low available licensed sessions. Returns a status of UP if the percentage of available licensed sessions is greater than the value specified for the Used licenses threshold property for the Sessions health indicator. Otherwise, returns a status of DOWN.
UMAliases	Checks that the Universal Messaging connection aliases for webMethods messaging. Returns a status of UP if all of the enabled Universal Messaging connection aliases are available. Otherwise, returns a status of DOWN.
Note:	Products installed on top of Microservices Runtime might provide their own health indicators.

Enabling and Disabling Health Indicators

Whether or not a health indicator is enabled determines if the healthy gauge includes the indicator when determining the UP or DOWN status of the Microservices Runtime. If you do not want the health gauge to include a particular indicator when determining the overall UP or DOWN status, disable the indicator. A disabled indicator does not execute when the health endpoint is invoked.

To enable or disable a health indicator

1. In the **Microservices** menu of the Navigation panel, click **Health Gauge**.
2. In the Health Indicator list, do one of the following:
 - To enable a disabled health indicator, click **No** in the **Enabled** column.
 - To disable an enabled health indicator, click **Yes** in the **Enabled** column.

Health Indicator Properties

Some of the health indicators have a configurable property that determines when a health indicator returns a status of UP or DOWN. For example, the ServiceThreads health indicator has the **Available threads threshold** which specifies the percentage of the

server threads that must be available for the indicator to return a status of UP. You can edit the threshold to tailor the indicator to your environment.

The following table identifies the configurable properties for the predefined health indicators.

Health Indicator	Property Name	Value
Diskspace	Free disk space threshold (as percentage of maximum available disk space)	Specify the percentage of free disk space out of the maximum available disk space above which the Diskspace health indicator returns a status of UP. When free disk space on the host or container on which Microservices Runtime resides is less than or equal to the specified percentage, the Diskspace health indicator returns a status of DOWN. The default is 10 percent.
Memory	Free memory threshold (as percentage of maximum memory)	Specify the percentage of free memory above which the Memory health indicator returns a status of UP. When free JVM memory for Microservices Runtime is less than or equal to the specified percentage, the Memory health indicator returns a status of DOWN. The default is 10 percent.
ServiceThread	Available threads threshold (as percentage of maximum server threads)	Specify the percentage of available server threads in the server thread pool at which the ServiceThread health indicator returns a status of UP. When the percentage of available threads is less than or equal to the specified percentage, the ServiceThread health indicator returns a status of DOWN. The default is 10 percent.
Sessions	Used licenses threshold (as percentage of total licensed sessions)	Specify the percentage of used licensed sessions at which the Sessions health indicator returns a status of DOWN. When the percentage of available licensed sessions is less than or equal to the specified percentage, the Sessions health indicator returns a status of DOWN. The default is 85 percent.

Configuring Health Indicator Properties

You can edit the properties of a health indicator to tailor the indicator for your environment. A health indicator with one or more configurable properties appears as a hypertext link in the Health Indicators List on the **Microservices > Health Gauge** page.

To configure health indicator properties

1. In the **Microservices** menu of the Navigation panel, click **Health Gauge**
2. In the Health Indicator list, click the name of the health indicator you want to configure.
3. On the **Microservices > Health Gauge > IndicatorName Properties** page, click Edit next to the property name.
4. On the **Microservices > Health Gauge > IndicatorName Properties > Edit** page, in the Value field, set a new threshold value for the property.
5. Click **Save Changes**.

Invoking the Health Gauge

You can invoke the health gauge via the `health` endpoint on the Microservices Runtime. When Microservices Runtime runs in a Docker container, you can use the `health` endpoint to monitor the state of the container from tools such as Kubernetes.

The request URL for the `health` endpoint is:

```
http://<hostname>:<port>/health
```

Where `<hostname>` is the IP address or name of the machine and `<port>` is the port number where Microservices Runtime is running.

Invocation of the `health` endpoint is restricted to users with Administrator access.

Note: The `health` endpoint is a predefined URL alias named “health” for the internal service that executes all enabled health indicators. Software AG does not recommend editing the predefined “health” URL alias. If you migrate to Microservices Runtime version 10.3 or higher from an earlier version and you already have a URL alias named “health”, Microservices Runtime does not create a health URL alias that points to the internal service. Any invocations of the `health` endpoint will not result in execution of health indicators. If you want to use the health gauge and the associated health indicators, you need to rename your existing health URL alias. Upon restart, Microservices Runtime creates a new health URL alias that corresponds to the `health` endpoint.

Obtaining Metrics for a Microservices Runtime

Microservices Runtime can generate metrics about the server and services on the server that the Prometheus server can use to provide insight to the operation of the Microservices Runtime and the services it contains. Microservices Runtime generates metrics in a Prometheus format. Prometheus is an open source monitoring and alerting toolkit which is frequently used for monitoring microservices.

Microservices Runtime exposes the metrics generating feature via the `metrics` endpoint. When the `metrics` endpoint is invoked, Microservices Runtime gathers server and service-level metrics and returns the data in a Prometheus format.

For a detailed list of the metrics returned by Microservices Runtime, see [“Prometheus Metrics Returned by Microservices Runtime” on page 78](#).

Note: The Microservices Runtime documentation assumes a familiarity with Prometheus technology. An in-depth discussion of Prometheus is beyond the scope of this guide but is available elsewhere.

Invoking the Metrics Endpoint

To instruct Microservices Runtime to gather metrics, you invoke the `metrics` endpoint on the Microservices Runtime. The request URL would be:

```
http://<hostname>:<port>/metrics
```

Where `<hostname>` is the IP address or name of the machine and `<port>` is the port number where Microservices Runtime is running.

Invocation of the `metrics` endpoint is restricted to users with Administrator access.

Note: The `metrics` endpoint is a predefined URL alias named “metrics” for the internal service that gathers statistics. Software AG does not recommend editing the predefined “metrics” URL alias. If you migrate to Microservices Runtime version 10.3 or higher from an earlier version and you already have a URL alias named “metrics”, Microservices Runtime does not create a `metrics` URL alias that points to the internal service. Any invocations of the `metrics` endpoint will not result in the gathering and return of metrics. If you want to use the metrics gathering functionality, you need to rename your existing metrics URL alias name. Upon restart, Microservices Runtime creates a new metrics URL alias that corresponds to the `metrics` endpoint.

8 Consul Support


■ Configuring Connections to Consul Server	64
■ Testing an Alias for the Consul Server	65
■ Setting the Default Alias for the Consul Server	66
■ Deleting a Consul Server Alias	66
■ Consul Public Services Folder	66

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.

<u>For this field</u>	<u>Specify</u>
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.

- 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


- Open Microservices Runtime Administrator.
- in the **Packages** menu of the Navigation panel, click **Management**.
- Click the  icon for the WmConsul package.
- On the Microservice Consul Registry Servers page, in the Microservice 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 table identifies the elements available in the `consul` folder of the WmConsul package:

Element	Package and Description
	WmConsul. De-registers a microservice from a Consul server. Use as a shutdown service for the package being registered as a microservice.
	WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul.
	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.
	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

<i>registryAlias</i>	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
<i>microserviceName</i>	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

<i>registryAlias</i>	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
<i>microserviceName</i>	String. Microservice for which you want to retrieve a list of active hosts.

Output Parameters

<i>hostList</i>	String List. An array containing the names of hosts of the microservice. Each element is in the format <i>host:port</i> (for example, <i>appserver.xyz.com:4555</i>). 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

<i>registryAlias</i>	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
<i>microserviceName</i>	String. Microservice for which you want to retrieve an active host.

Output Parameters

<i>hostName</i>	String. Name of a host of the microservice, chosen randomly. The host name is in the format: <i>port</i> (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

<i>registryAlias</i>	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.
<i>microserviceName</i>	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 Runtime configuration only. That is, a configuration variables template does not include key-value pairs for all of the configuration information for a Microservices Runtime. Only some configuration information is supported for use with 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	<code>cachemanager.cacheManagerName.urls</code>
Email Port	Host Name	<code>email.packageName.aliasName.host</code>
	Password	<code>email.packageName.aliasName.password</code>
	Port	<code>email.packageName.aliasName.server_port</code>
	User Name	<code>email.packageName.aliasName.user</code>
File Polling Port	Completion Directory	<code>filepolling.packageName.aliasName.completionDir</code>
	Error Directory	<code>filepolling.packageName.aliasName.errorDir</code>
	Monitoring Directory	<code>filepolling.packageName.aliasName.monitorDir</code>
	Working Directory	<code>filepolling.packageName.aliasName.workDir</code>
Global Variables	Value	<code>globalvariable.globalVariableName.value</code>

Asset	Property	Key Name
JDBC Pool Alias	Database URL	<code>jdbc.aliasName.dbURL</code>
	User ID	<code>jdbc.aliasName.userid</code>
	Password	<code>jdbc.aliasName.password</code>
JMS Connection Alias	Connection Client ID	<code>jms.aliasName.clientID</code>
	User	<code>jms.aliasName.user</code>
	Password	<code>jms.aliasName.password</code>
JNDI Provider Alias	Provider URL	<code>jndi.aliasName.providerURL</code>
	Provider URL Failover List	<code>jndi.aliasName.providerURLFailoverList</code>
	Security Principal	<code>jndi.aliasName.securityPrincipal</code>
	Security Credentials	<code>jndi.aliasName.securityCredentials</code>
Kerberos	Realm	<code>kerberos.KerberosConfig.kerberosConfig</code>
	Key Distribution Center Host	<code>kerberos.KerberosConfig.kerberosKDC</code>
	Kerberos Configuration File	<code>kerberos.KerberosConfig.kerberosRealm</code>
	Use Subject Credentials Only	<code>kerberos.KerberosConfig.useSubjectCreds</code>
Keystore Alias	Location	<code>keystore.aliasName.ksLocation</code>

Asset	Property	Key Name
	Password	<code>keystore.aliasName.ksPassword</code>
	Key Alias Password	<code>keystore.aliasName.keyAlias.keyAliasName.keyAliasPassword</code>
LDAP Configuration	Directory URL	<code>ldap.ldapDirectory.url</code>
	Principal	<code>ldap.ldapDirectory.prin</code>
	Credentials	<code>ldap.ldapDirectory.password</code>
Proxy Server Alias	Host Name or IP Address	<code>proxyserver.aliasName.host</code>
	Port Number	<code>proxyserver.aliasName.port</code>
	User Name	<code>proxyserver.aliasName.username</code>
	Password	<code>proxyserver.aliasName.password</code>
Proxy Server Bypass	Addresses	<code>settings.watt.net.proxySkipList</code>
Remote Server Alias	Host Name or IP Address	<code>remoteserver.aliasName.host</code>
	Port Number	<code>remoteserver.aliasName.port</code>
	User Name	<code>remoteserver.aliasName.user</code>
	Password	<code>remoteserver.aliasName.password</code>
	Retry Server	<code>remoteserver.aliasName.retryServer</code>
Server configuration parameters (server.cnf)	Any	<code>settings.serverConfigurationParameterName</code>

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

Asset	Property	Key Name
	Port Number (HTTP/S Transport Properties)	<code>webserviceendpoint.type.protocol. aliasName.transportInfo.port</code>
	User Name (HTTPS Transport Properties)	<code>webserviceendpoint.type.protocol. aliasName.transportInfo.user</code>
	Password (HTTPS Transport Properties)	<code>webserviceendpoint.consumer.protocol. aliasName.transportInfo.transportPassword</code>
	User Name (WS Security Properties)	<code>webserviceendpoint.type.protocol. aliasName.messageInfo.user</code>
	Password (WS Security Properties)	<code>webserviceendpoint.type.protocol. aliasName.messageInfo.messagePassword</code>
	To (Message Addressing Properties)	<code>webserviceendpoint.type.protocol. aliasName.messageaddressingproperties. toMsgAddr</code>
	From (Message Addressing Properties)	<code>webserviceendpoint.type.protocol. aliasName.messageaddressingproperties. fromMsgAddr</code>
	Reply To (Message Addressing Properties)	<code>webserviceendpoint.type.protocol. aliasName.messageaddressingproperties. replyToMsgAddr</code>
	Fault To (Message Addressing Properties)	<code>webserviceendpoint.type.protocol.aliasName. messageaddressingproperties.faultToMsgAddr</code>

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

B Prometheus Metrics

■ Prometheus Metrics Returned by Microservices Runtime	78
■ Server Metrics	78
■ Service Metrics	80

Prometheus Metrics Returned by Microservices Runtime

When the `metrics` endpoint is called, Microservices Runtime gathers metrics and returns the data in a Prometheus format. Prometheus is an open source monitoring and alerting toolkit which is frequently used for monitoring containers.

Microservices Runtime returns server-level metrics and service-level metrics. All of the metrics returned by Microservices Runtime are gauge metrics, one of the core metric types. The following sections identify the server and service-level metrics gathered and returned by Microservices Runtime.

Note: The Microservices Runtime documentation assumes a familiarity with Prometheus technology. An in-depth discussion of Prometheus is beyond the scope of this guide but is available elsewhere.

Server Metrics

Server metrics are metrics about the Microservices Runtime. All server metrics have a Prometheus label of `host=<hostname|ipaddress>`.

The following table identifies the server metrics returned by Microservices Runtime when the `metrics` endpoint is called.

Prometheus Metric Name	Description
<code>sag_is_max_memory_bytes</code>	The maximum memory of the Microservices Runtime JVM.
<code>sag_is_total_memory_bytes</code>	The total memory for the Microservices Runtime JVM.
<code>sag_is_free_memory_bytes</code>	The total free memory for the Microservices Runtime JVM.
<code>sag_is_used_memory_bytes</code>	The total used memory for the Microservices Runtime JVM.
<code>sag_is_threads</code>	The total number of threads in use on the Microservices Runtime.
<code>sag_is_max_threads</code>	The maximum number of threads in the server thread pool.

Prometheus Metric Name	Description
<code>sag_is_service_threads</code>	The total number of threads used for service execution.
<code>sag_is_peak_service_threads</code>	The peak number of threads used by the server since Microservices Runtime startup.
<code>sag_is_system_threads</code>	The total number of system threads which is any thread not obtained from the server thread pool.
<code>sag_is_max_system_threads</code>	The peak number of system threads used by Microservices Runtime since startup.
<code>sag_is_number_current_connections</code>	The number of current connections to the Microservices Runtime.
<code>sag_is_max_connections</code>	The peak number of connections to Microservices Runtime.
<code>sag_is_average_connection_time</code>	The running average for the connection time expressed in milliseconds.
<code>sag_is_http_requests</code>	The total number of HTTP/S requests since the last statistics poll. The statistics poll interval is controlled by the <code>watt.server.stats.pollTime</code> server configuration parameter. The default interval is 60 seconds.
<code>sag_is_total_http_requests</code>	The total number of HTTP/S requests since Microservices Runtime startup.
<code>sag_is_avg_time_per_http_request</code>	The running average time for all HTTP/S requests measured in milliseconds.
<code>sag_is_service_starts_per_minute</code>	The number of top-level services started each minute.
<code>sag_is_service_completions_per_minute</code>	The number of top-level service completions per minute.
<code>sag_is_number_unknown_errors</code>	The number of errors that were caused by exceptions that are not <code>ServiceExceptions</code> .

Prometheus Metric Name	Description
<code>sag_is_number_sessions_used</code>	The number of sessions used since Microservices Runtime startup.
<code>sag_is_peak_number_sessions</code>	The peak number of sessions since Microservices Runtime startup.
<code>sag_is_max_number_sessions_allowed</code>	The maximum number of sessions allowed at one time as determined by the license.
<code>sag_is_current_stateful_sessions</code>	The current number of stateful sessions.
<code>sag_is_peak_stateful_sessions</code>	The peak number of stateful sessions since Microservices Runtime startup.
<code>sag_is_stateful_sessions</code>	The maximum number of stateful sessions. If <code>watt.server.session.stateful.max</code> is set to 0 or <code>watt.server.session.stateful.enableLimit</code> is set to false, this is the maximum number of sessions as specified in the license. If <code>watt.server.session.stateful.max</code> is not 0 and <code>watt.server.session.stateful.enableLimit</code> is set to true, then this is the value of <code>watt.server.session.stateful.max</code> .

Service Metrics

Service metrics are metrics about a service on Microservices Runtime. When the metrics endpoint is called, Microservices Runtime returns service metrics for services called as top-level services. Microservices Runtime does not return metrics for internal services or nested services which are those service invoked within another service.

Note: A top-level service is a service that is invoked by a client request.

All service metrics have a Prometheus label of `service="<serviceName>"`

The following table identifies the service metrics returned by Microservices Runtime when the `metrics` endpoint is called.

Prometheus Metric Name	Description
<code>sag_is_service_number_access</code>	The number of times the service has executed since the Microservices Runtime last started.
<code>sag_is_service_number_running</code>	The number of concurrent service executions for this service at Microservices Runtime gathered the metrics.
<code>sag_is_service_number_cache_hits</code>	The total number of times that a cached service result pipeline has been returned instead of calling the service since the cache was last reset.
<code>sag_is_service_cache_hit_ratio</code>	The percentage of request for the service that have been fulfilled using cached service results since the cache was last reset.
<code>sag_is_service_cache_entries</code>	The current number of active cached entries for this service in service results cache.
<code>sag_is_service_cache_expires</code>	The current number of expired cache entries for the service.
<code>sag_is_service_number_prefetches</code>	The number of times that Microservices Runtime has prefetched and then cached the results for this service since the cache was last reset.
<code>sag_is_service_number_recent_prefetches</code>	The number of times Microservices Runtime has prefetched and cached the results for the service since the last statistics poll. The statistics poll interval is controlled by the <code>watt.server.stats.pollTime</code> server configuration parameter. The default interval is 60 seconds.
<code>sag_is_service_number_circuit_breaker</code>	The number of times the circuit has opened for this service since Microservices Runtime last started. This metric applies only to services for which a circuit breaker is enabled.
<code>sag_is_service_number_circuit_breaker_requests</code>	The number of requests for the service when the circuit was in an open state since Microservices Runtime last started. This metric

Prometheus Metric Name	Description
	applies only to services for which a circuit breaker is enabled.
