webMethods Microgateway User's Guide

Version 10.3

October 2018

WEBMETHODS

This document applies to webMethods Microgateway 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.

Document ID: YAM-UG-103-20200224

Table of Contents

About this Guide **5**
Document Conventions 6
Online Information and Support 6
Data Protection 7

1 webMethods Microgateway **9**
Introduction 10
API Gateway Integration 11
Microgateway Installation 11

2 Asset and Configuration Provisioning **13**
Asset provisioning 14
Configuration Provisioning 17

3 Microgateway Provisioning **21**
Microgateway Provisioning 22
Instance-based Provisioning 22
Docker-based Provisioning 24

4 Policies **27**
Policies Supported in Microgateway 28
Transport 28
Identify and Access 29
Routing 34
Traffic Monitoring 35
Error Handling 37

5 Publishing APIs to Runtime Service Registries **41**
Publishing APIs to Runtime Service Registries 42

6 Command Line Reference **45**
Microgateway Command Line Reference 46

About this Guide

- Document Conventions 6
- Online Information and Support 6
- Data Protection 7

This guide describes how you can use API Gateway and other API Gateway components to effectively manage APIs for services that you want to expose to applications, whether inside your organization or outside to partners and third parties.

To use this guide effectively, you should have an understanding of the APIs that you want to expose to the developer community and the access privileges you want to impose on those APIs.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	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.
...	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 webMethods Microgateway

- Introduction 10
- API Gateway Integration 11
- Microgateway Installation 11

Introduction

The adoption of the micro-service architecture pattern drives the need for lightweight gateways or Microgateways. The Microgateway gives control over a micro-service landscape by enforcing policies which perform authentication, traffic monitoring, and traffic management. The lightweight nature of a Microgateway allows a flexible deployment to avoid gaps or bottlenecks in the policy enforcement.

Microgateway is a gateway that enables micro-services to communicate with each other directly without re-routing the communication channel through an API Gateway. This eases out the traffic overload on API Gateway with communication between micro-services. You can enforce the required protection policies on the Microgateway to have a secure communication channel between the micro-services.

This figure illustrates an API Gateway with micro-services suffering from network latency and security issues as figure 1 followed by figure 2 that depicts a Microgateway with micro-services addressing network latency and security issues by side car deployment.

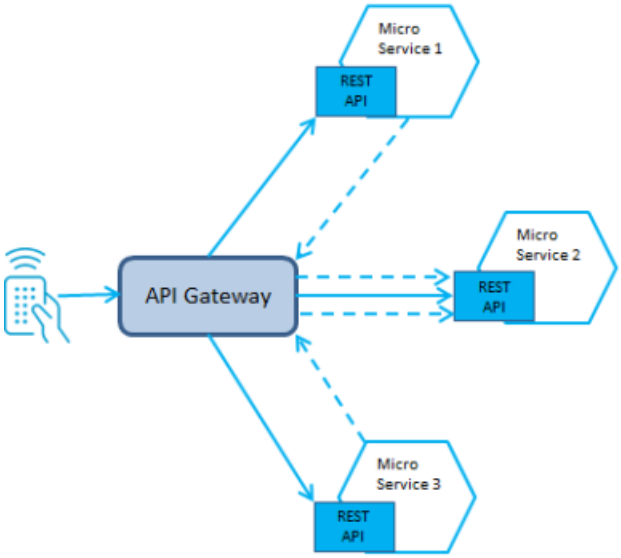


Fig1: API Gateway with micro services suffering from network latency and security issues

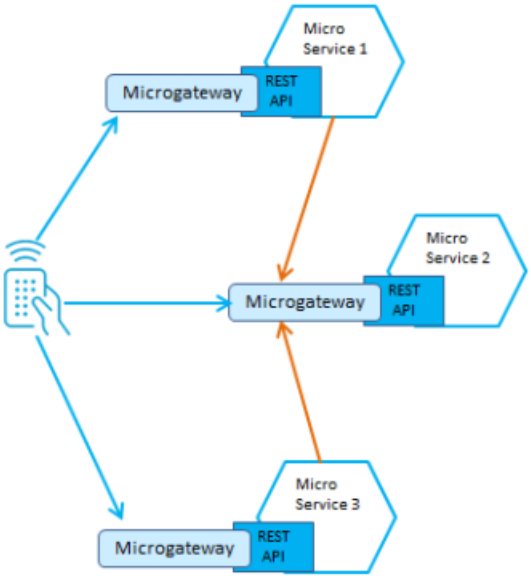


Fig 2: Microgateway with micro services addressing network latency and security issues by side car deployment

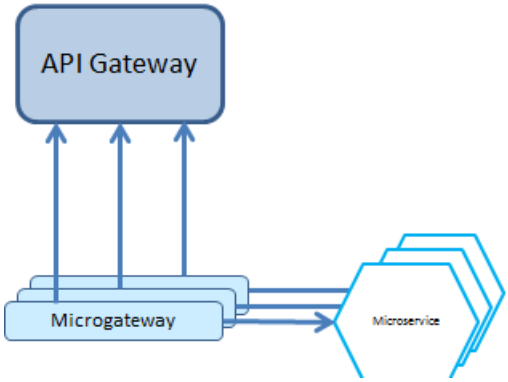
The first part, figure 1, of the diagram depicts micro-services with a single API Gateway that does the enforcement of policies. Here, each micro-service exposes an endpoint where no policy enforcement is done. Moreover, considering that the micro-services are interacting with each other, all the traffic needs to be routed to the API Gateway. This leads to additional network latency and the API Gateway might become a bottleneck. In the second part, figure 2, API Gateway is replaced with a set of Microgateways that are deployed near the micro-services. Such a sidecar deployment does not leave any gaps and avoids bottlenecks, thereby solving the network latency issues and ease of policy enforcement.

Microgateway Components

Microgateway comes with a service performing the policy enforcement on REST APIs. The Microgateway service runs within its own Java runtime environment and is controlled by a simple command line interface that supports basic lifecycle operations like start and stop. The configuration of the service consists of system settings and assets that can be provisioned from a running API Gateway or can be provisioned through a filesystem. The provisioned assets include application, API, policy, and alias definitions. The Microgateway service exposes an administrator REST API to query the status, the system setting, and the provisioned assets and shut down microgateway as well.

API Gateway Integration

The Microgateway's responsibility is focused on a single micro-service or a small number of micro-services. To manage a micro-service landscape an API Gateway is required. It offers the user interface for configuring the policy configuration and system configurations. Moreover, it is responsible for monitoring the traffic within the micro-service landscape. The following figure shows how Microgateways are interacting with an API Gateway.



The Microgateways pull the assets including APIs, applications, and policies from the API Gateway where they are configured. Also, other administrative settings including SSL configuration and fault configurations are defined in the API Gateway and pulled by the Microgateway during startup. The download of assets is done through the API Gateway REST APIs.

While monitoring the API requests and responses, the Microgateway pushes the runtime analytics information to API Gateway. API Gateway provides a consolidated view through its dashboards.

The download of assets and administrative settings can be done up-front using the Microgateway tooling that allows to provision stand-alone Microgateways, which do not require any connection to a running API Gateway. Stand-alone Microgateways do not allow to perform a consolidated traffic monitoring of a micro-service landscape.

Microgateway Installation

You can install a Microgateway using Software AG Installer. In the Software AG Installer, Microgateway appears as a subnode under the API Gateway node in the installer tree. Microgateway

is selected by default when you select API Gateway in the installer tree. For details on installing using installer, see *Installing Software AG Products*.

Note:
Microgateway can only be installed together with the API Gateway. An independent installation of the Microgateway is not supported. The Microgateway installation becomes operational only if the Microgateway feature is active in the API Gateway license.

2 Asset and Configuration Provisioning

- Asset provisioning 14
- Configuration Provisioning 17

Asset provisioning

Provisioning assets in Microgateway makes assets available for use in Microgateway. The assets you can provision in Microgateway are, APIs including policies and policy properties, runtime aliases, applications, and global policies.

Note:
Microgateway only supports the REST APIs and do not support the SOAP and OData APIs.

Asset provisioning in Microgateway covers the following:

- Reading APIs, policies and applications from API Gateway on Microgateway start
- Updating APIs and policies from API Gateway
- Updating Applications from API Gateway
- Reading APIs, policies, and applications from a file system by exporting an archive
- Reading administrative settings from API Gateway

You can provision assets to a Microgateway in one of the following ways:

- API Gateway export archive-based provisioning
- Pulling assets from API Gateway

Microgateway also supports a mixed provisioning. If case of same API names the API Gateway archives are preferred over the assets being pulled from API Gateway.

Asset Provisioning through API Gateway Export Archive

You can start the Microgateway server with one or more API Gateway export archives that contain the assets to be provisioned. Microgateway supports export archives of version 10.3 or higher. All the supported assets are imported during Microgateway startup.

You can pass on the API Gateway export archive in the start command through the archive parameter:

```
microgateway.cmd start -p 9090 --archive apigw_archive.zip
```

You can specify multiple archives using a comma separated list. Ensure that there is no space within the comma separated list.

```
microgateway.cmd start -p 9090
--archive apigw_archive1.zip,apigw_archive2.zip
```

The Microgateway reads the archives in the specified order. If there are any assets provisioned by an earlier archive, they are overwritten by the assets present in the later archives.

Creating API Gateway Archives using Command Line

You can use the `createAssetArchive` command that Microgateway CLI provides for reading an archive. The command has the following parameters:

- `api_gateway`: API Gateway url
- `api_gateway_user`: API Gateway user
- `api_gateway_password`: API Gateway password
- `apis`: List of APIs, comma separated, specified using name, identifier or name and version combination (name and version are separated using /). You can use the wildcard character * for API name.
- `policies`: List of global policies, comma separated, specified using name or identifier
- `applications`: List of global applications, comma separated, specified using name or identifier.
- `archive`: Archive to create.

A Sample workflow for Asset Provisioning using import package

A sample sequence for asset provisioning by using import package looks as follows. A sample archive BayernAndEmployeeService.zip is used to describe the example.

1. Start API Gateway with given archive BayernAndEmployeeService.zip

   ```
   microgateway.bat start -p 9090 -a
   "C:\Users\tfi\Work\API-Gateway\ExportArchives\10.3\BayernAndEmployeeService.zip"
   ```

2. Check status.

   ```
   GET http://localhost:9090/rest/microgateway/status
   ```

   The status response looks like this:

   ```
   {
       "description": "webMethods Microgateway",
       "publisher": "Software AG",
       "version": ""
   }
   ```

3. For fetching details about the deployed APIs, applications and global policies send the following request.

   ```
   GET http://localhost:9090/rest/microgateway/assets
   ```

   The details show the provisioned APIs.

4. Invoke the Employee API.

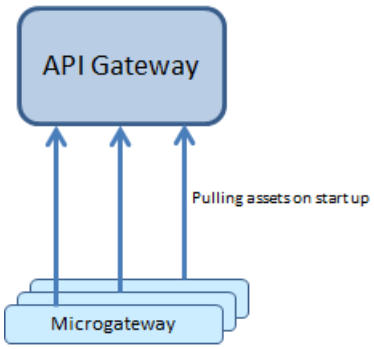
   ```
   GET http://localhost:9090/gateway/EmployeeService/employees
   ```

5. Stop the Microgateway listening on port 9090.

```
microgateway.bat stop -p 9090
```

Pulling Assets from API Gateway

You can start the Microgateway server with a URL and credentials pointing to an API Gateway. Microgateway pulls the assets from the referenced API Gateway.



Microgateway can only pull assets from API Gateways with version 10.3 or higher. Multiple Microgateways can pull assets from the same API Gateway. To pull assets from API Gateway, Microgateway start command offers the following options:

- api_gateway: API Gateway url
- api_gateway_user:API Gateway user
- api_gateway_password: API Gateway password
- apis: List of APIs, comma separated, specified using name, identifier or name and version combination (name and version are separated using /). You can use the wildcard character * for API name.
- policies: List of global policies, comma separated, specified using name or identifier
- applications: List of global applications, comma separated, specified using name, identifier or name and version combination (name and version are separated using /)

Connecting to API Gateway

Microgateway connects to the API Gateway during startup. If the API Gateway can't be contacted, then Microgateway terminates with an error message as follows:

```
Can't connect to API Gateway, going down ...
Host not reachable.
Please check your Microgateway configuration.
```

Pulling Specific Assets

An API, policy or application is identified either using its unique identifier or by the combination of name and version. If an asset name or identifier can't be resolved, a respective error message is written to the Microgateway log.

For an identified API, the API along with the API-level policies and policy properties, registered applications, and referenced runtime aliases are pulled from API Gateway.

For an identified global policy, the policy, the policy properties, and referenced runtime aliases are pulled from API Gateway.

For an identified global application, only the application is pulled. For runtime aliases, the default values become effective.

Pulling All Assets

If you do not specify an API, application or policy, then no assets are pulled from the API Gateway.

- If an API has a unsupported policy, the provisioning of the API is rejected.
- If a global policy references an unsupported policy, then import of the global policy is rejected.
- When provisioning runtime aliases the default values become effective.

A Sample workflow for Asset Provisioning by pulling API Gateway

A sample sequence for asset provisioning by pulling API Gateway looks as follows. A sample imported file BayernAndEmployeeService.zip is used to describe the example.

1. Start an API Gateway and import the BayernAndEmployeeService.zip. The import results into 2 APIs - EmployeeService and BayernRest
2. Start Microgateway with given a API Gateway URL and API name.

```
microgateway.bat start -p 9090
-gw http://localhost:5555/rest/apigateway -gwu Administrator -gwp manage
-apis EmployeeService
```

3. Check deployed assets .

```
GET http://localhost:9090/rest/microgateway/assets
```

4. Calling API.

```
GET http://localhost:9090/gateway/EmployeeService/employees
```

5. Stop the Microgateway listening on port 9090 .

```
microgateway.bat stop -p 9090
```

6. Start Microgateway with given a API Gateway URL with and multiple API names .

```
microgateway.bat start -p 9090
-gw http://localhost:5555/rest/apigateway -gwu Administrator
-gwp manage -apis EmployeeService,BayernRest
```

Configuration Provisioning

Microgateway works with a system configuration file, which is installed with default settings under config/system-settings.yml. The file contains the following entries:

- `faults`: contains variables for error handling during runtime.

- `extended_settings`: various kinds of settings for runtime.
- `gateway_destination`: API Gateway settings for logging into API Gateway.
- `key_store`: Keystore settings for establishing https connections.
- `trust_store`: Truststore settings for HTTPS handshake for specific policies.
- `system`: internal settings.

Configuration is part of the export archive or pulled from API Gateway.

When you start Microgateway with an archive (either directly or by pulling from a running API Gateway) the settings are incorporated and used.

By default all settings are picked up from the export archive or pulled from API Gateway and the default configuration settings in system-settings.yml are overwritten. To avoid overwriting the default configuration the `retain-settings` option can be specified when starting Microgateway. When you specify the `retain_settings` option the specified and non-empty parameters from system-settings.yml are used instead of those from the export archive or the settings pulled from API Gateway.

To enable the policy enforcement on a Microgateway the following configurations need to be provisioned in Microgateway:

- extended
- apiFault
- elasticsearchDestinationConfig
- gatewayDestinationConfig

Note:
The external Elasticsearch configuration is optional, and needs to be specified if you have such an Elasticsearch in your environment. You require an external Elasticsearch if you have the Log Invocation policy enforced where Elasticsearch destination is selected.

The default configuration is available in the config/system_settings.yml:

```
---
faults:
   default_error_message: "API Gateway encountered an error.
   Error Message: $ERROR_MESSAGE. Request Details: Service - $SERVICE,
   Operation - $OPERATION, Invocation Time:$TIME, Date:$DATE,
   Client IP - $CLIENT_IP, User - $USER and Application:$CONSUMER_APPLICATION"
   native_provider_fault: "false"
extended_settings:
   defaultEncoding: "UTF-8"
   apiKeyHeader: "x-Gateway-APIKey"
   apig_MENConfiguration_tickInterval: "60"
   events.collectionQueue.size: "10000"
   events.collectionPool.minThreads: "1"
   events.collectionPool.maxThreads: "8"
gateway_destination:
   sendPolicyViolationEvent: "true"
```

```
key_store:
   type: JKS
   provider: SUN
   location: config/keystore.jks
   password: manage
system:
   version: "@full_version@"
---
```

3 Microgateway Provisioning

- Microgateway Provisioning 22
- Instance-based Provisioning 22
- Docker-based Provisioning 24

Microgateway Provisioning

Microgateway provisioning allows you to spawn multiple Microgateway instances from a single Microgateway installation. The instances can be defined as self-contained including assets and configuration. You can only provision a pre-configured Microgateway.

You can provision a Microgateway in one of the following ways:

- Instance-based provisioning.
- Docker-based provisioning.

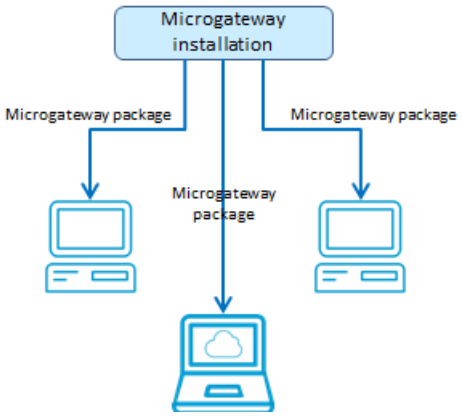
Instance-based Provisioning

You can create a Microgateway package from an existing installation and copy it to multiple target machines. A Microgateway instance package is a self-contained zip file that contains all artifacts for running a Microgateway. The contents of the zip file are:

- JRE
- Microgateway server
- Microgateway cli
- Configuration files
- One or more API Gateway export archives

After copying the zip file , you have to extract the zip file contents and run the commandline scripts of the Microgateway CLI to start the Microgateway. A package is configured based on a Microgateway configuration file. The configuration file either points to an API Gateway or to one or more API Gateway export archives for asset promotion. The asset promotion is performed when you start the Microgateway within the package.

The figure illustrates creating a Microgateway package from an existing installation and copying it to multiple target machines.



Creating a Microgateway Instance

For the package-based provisioning the Microgateway CLI provides the `createInstance` command. The command creates a Microgateway package that can be transferred to any target environment. The command has the following parameters:

- `config`: optional Microgateway configuration file
- `instance`: optional name of the Microgateway package. The default value is MicrogatewayInstance.zip

Instance-based Provisioning Flow

A sample sequence for executing an instance-based provisioning is as follows:

1. Create API Gateway archives and set the configuration file accordingly.

```
---
ports:
   http: 5554
   https: 5553

api_gateway:
   url: http://<host>:<port>/rest/apigateway
   user: Administrator
   password: <pwd>
archive:
   file: export.zip


---
```

2. Create a Microgateway instance.

```
microgateway.bat createInstance --config config.yml
--instance c:\temp\Microgateway.zip -os win
```

   The provided config.yml is added to the MyMicrogateway.zip and overwrites any already given configuration file.

3. Copy the Microgateway instance to the target environment.

4. Extract the contents of the zip file using the unzip command and start the Microgateway instance.

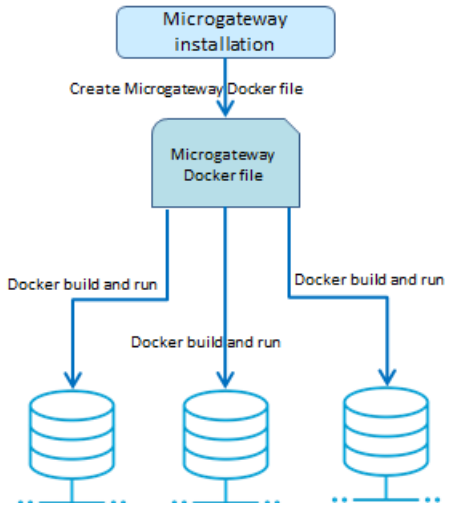
```
unzip MyMicrogateway.zip .
./microgateway.bat start --config config.yml
```

   The unzip operation creates the sub-folders of the Microgateway. The start command picks up the config file from the default location (config/config.yml). Ensure that on the following files the execute-bit is set by applying the chmod command (linux-only):

   - microgateway.sh
   - microgateway-jre-linux/bin/java

Docker-based Provisioning

Docker based provisioning involves the creation of a Microgateway Docker file from an existing installation, building the image, and running it multiple times in a container environment as depicted in the following figure.



Microgateway Docker image

For the Docker-based provisioning the Microgateway CLI provides the `createDockerFile` command. The command creates a Docker file that can be consumed by docker build for creating a Docker image. The Microgateway Docker image contains an unzipped Microgateway package.

The command contains the following parameters:

- docker_from: optional base image, for adding a Microgateway layer on top of an existing image. If no base image is given, the default is alpine:3.7 (4.1.MB).
- docker_dir: Microgateway directory
- docker_file: optional docker file name. Default is Microgateway_DockerFile

Dynamic configuration is achieved through volume mapping. The config subfolder can be mapped.

You can run the Docker image to spawn a Docker container from the created docker image.

A Sample workflow for Docker-based Provisioning with Export Archive

A sample sequence for executing a Docker-based provisioning with export archive looks as follows.

1. Create API Gateway archives and set the configuration file accordingly

```
---
ports:
   http: 5554
   https: 5553

api_gateway:
```

```
   url: http://<host>:<port>/rest/apigateway
   user: Administrator
   password: <pwd>

archive:
   file: export.zip

---
```

2. Create Microgateway Docker file.

```
./microgateway.sh createDockerFile --docker_dir . -p 9090 -a <file-name>.zip
```

   The command creates the Docker file Microgateway_DockerFile that copies the export archives referenced by the zip file into the Docker image.

   The createDockerFile command allows to configure the Docker file creation through command line parameters. For example, the following command sequence creates a Docker file for a Microgateway container listening on port 9090 and with the assets from the archives apis.zip and policies.zip.

```
microgateway createDockerFile -p 9090 -c config.yml
-a apis.zip,policies.zip
```

   The command line parameters take precedence over the configuration values specified in the config.yml.

3. Create the Docker image with asset archive.

```
docker build -t sag:mcgw-static -f Microgateway_DockerFile
```

   The command creates the image sag:mcgw from the generated Docker file.

4. Run the Docker image.

```
docker run -d -p 9090:9090 --name mcgw-static sag:mcgw-static
```

   The command spawns a Docker container from the image sag:mcgw. The contained Microgateway listens at the host port 9090.

Dynamic configuration is applied by mapping a volume holding the config folder of the Microgateway instance.

A Sample workflow for Docker-based Provisioning with pulling from API Gateway

A sample sequence for executing a Docker-based provisioning with pulling from API Gateway looks as follows.

1. Create Microgateway Docker file points to an API Gateway for pulling APIs on startup.

```
./microgateway.sh createDockerFile --docker_dir . -c config/config.yml
```

   The command creates the Docker file Microgateway_DockerFile that copies the API Gateway configurations from the config.yml file into the Docker image.

2. Create the Docker image.

```
docker build -t sag:mcgw-dynamic -f Microgateway_DockerFile
```

   The command creates the image sag:mcgw from the generated Docker file.

3. Run the Docker image.

```
docker run -d -p 9090:9090 --name mcgw-dynamic sag:mcgw-dynamic
```

   The command spawns a Docker container from the image sag:mcgw. The contained Microgateway listens at the host port 9090.

A Sample workflow for Docker-based Provisioning with dynamic configuration

A sample sequence for executing a Docker-based provisioning with dynamic configuration looks as follows.

1. Create Docker file pointing to the config.yml file.

```
./microgateway.sh createDockerFile --docker_dir . -c config/config.yml -p 9090
```

   The command creates the Docker file Microgateway_DockerFile that copies the configurations from the config.yml file into the Docker image.

2. Create the Docker image.

```
docker build -t sag:mcgw-dynamic -f Microgateway_DockerFile
```

   The command creates the image sag:mcgw from the generated Docker file.

3. Provide config directory with host file system with required Microgateway configurations:

```
> ls ~/mcgw-conf/
config.yml  keystore.jks  license.xml  system-settings.yml
```

4. Start container with volume mapping pointing to config directory in host file system.

```
docker run -d -v ~/mcgw-conf:/Microgateway/config
-p 9090:9090 --name mcgw-dynamic sag:mcgw-dynamic
```

   The command spawns a Docker container from the image sag:mcgw. The contained Microgateway listens at the host port 9090.

4 Policies

- Policies Supported in Microgateway 28
- Transport 28
- Identify and Access 29
- Routing 34
- Traffic Monitoring 35
- Error Handling 37

Policies Supported in Microgateway

This section provides information about the runtime policies supported in Microgateway. A policy can be enforced on an API to perform specific tasks, such as transport, authorization, routing of requests to target services, logging, , and error handling of data. For example, a policy could instruct Microgateway to perform any of the following tasks and prevent malicious attacks:

- Verify that the requests submitted to an API come from applications that are authenticated and authorized using only Basic Auth and API Key headers.
- Limits the number of invocations during a specified time interval for a particular API and for applications, and send alerts to API Gateway when these performance conditions are violated.
- Log the request and response messages.

Note:
These policies are configured in API Gateway and provisioned to Microgateway. Microgateway neglects the configurations that are supported.

Policies are grouped into stages as per their usage. For example, the policies in the Identify and Access stage can be enforced on an API to specify the kind of identifiers that are used to identify the application and authorize it against all applications registered in Microgateway.

Microgateway provides the following system-defined policies that are grouped into stages depending on their usage

- Transport policy
  - Enable HTTP / HTTPS
- Identity & Access
  - Identify & Authorize Application (Basic, API Key)
- Routing
  - Straight Through Routing
- Traffic Monitoring
  - Log Invocation
  - Throttling Traffic Optimization
- Error Handling
  - Conditional Error Processing

Transport

The policies in this stage specify the protocol to be used for an incoming request during communication between Microgateway and an application. The policy included in this stage is Enable HTTP/HTTPS.

Enable HTTP/HTTPS

This policy specifies the protocol to use for an incoming request to the API on Microgateway. If you have a native API that requires clients to communicate with the server using the HTTP and HTTPS protocols, you can use the Enable HTTP or HTTPS policy. This policy allows you to bridge the transport protocols between the client and Microgateway.

For example, you have a native API that is exposed over HTTPS and an API that receives requests over HTTP. If you want to expose the API to the consumers of Microgateway through HTTP, then you configure the incoming protocol as HTTP.

The table lists the properties that you can specify for this policy:

Parameter	Description
Protocol	Specifies the protocol (HTTP or HTTPS) to be used to accept and process the requests. The following properties when set specify the following: ■ **HTTP**. Microgateway accepts requests that are sent using the HTTP protocol. This is the default setting. ■ **HTTPS**. Microgateway accepts requests that are sent using the HTTPS protocol.

Identify and Access

The policy in this stage provides different ways of identifying and authorizing the application, and provides the required access rights for the application.Microgateway supports the following Identify and access management policy:

- Identify & Authorize Application (Basic, API Key)

The Identify and Authorize Application policy is used to identify the application, authenticate the request based on policy configured and authorizes it against all applications registered in API Gateway.

Identify and Authorize Application

This policy authorizes and allows access to the applications that are trying to access the APIs, for example, through IP address or hostname, and validate the clients credentials.

The table lists the properties that you can specify for this policy:

Parameter	Description
Condition	Specifies the condition operator for the identification and authentication types.

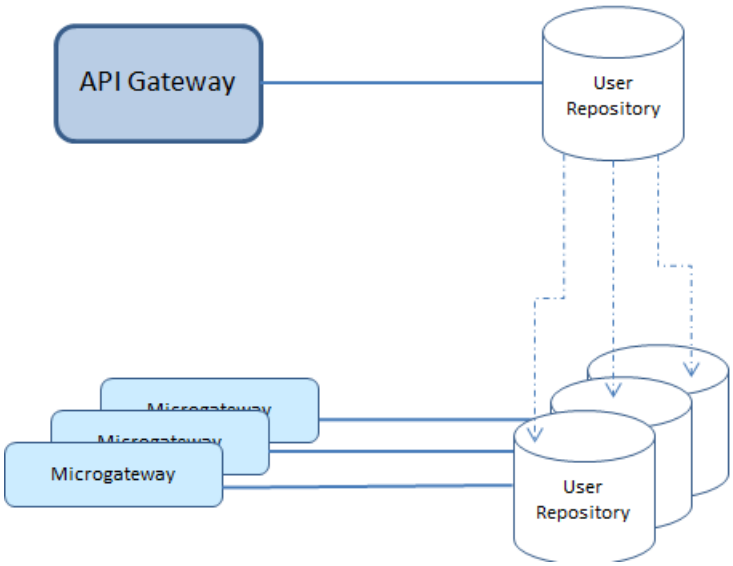
Parameter	Description
	You can specify any of the following condition operators: ■ **AND**. Applies all the identification and authentication types. ■ **OR**. Applies one of the selected identification and authentication types. **Note:** Even though this policy provides the option of choosing an **AND** or **OR** operation between the different identification and authentication types, the operation across the different policies in the IAM stage is always **AND**. For example, configuring the Identify and Authorize Application policy with **API Key** and the Inbound Authentication - Transport policy with **HTTP Basic Authentication** using an **OR** operation is not supported.
Allow anonymous	Specifies whether to allow all users to access the API without restriction. When you add a security policy and configure **Allow anonymous**, all requests are allowed to pass through to the native API, but the successfully identified requests are grouped under the respective identified application, and all unidentified requests are grouped under a common application named `unknown`. While you allow all requests to pass through you can perform all application-specific actions, such as, viewing the runtime events for a particular application, monitor the service level agreement for a few applications and send an alert email based on some criteria like request count or availability, and throttle the requests from a particular application and not allow the request from that application if the number of requests reach the configured hard limit within configured period of time.
Identification Type. Specifies the identification type. You can select any of the following identification types.	
API Key	Specifies using the API key to identify and validate the client's API key to verify the client's identity in the registered list of applications for the specified API.
HTTP Basic Authentication	Specifies using Authorization Header in the request to identify and authorize the client application against the list of applications with the identifier `username` in Microgateway. Provide one of the following **Application Lookup condition**: ■ **Registered applications**. Tries to verify the client's credentials against the list of registered applications for the specified API.

Parameter	Description
	■ **Global applications**. Tries to verify the client's credentials against a list of all global applications available in Microgateway. ■ **Do not identify**. Checks for the existence of the criterion but does not validate if the specified value is a valid application and forwards the request to the native API. For example, HTTP Basic Authentication is checked by the HTTP transport level property Authorization: Basic Base64encodesusernamepassword

User Identification to Support Identity and Access Management Policy

Microgateway supports authentication against users that are defined through API Gateway. The authentication is performed against a read-only user repository. This ensures that users can be authenticated even if Microgateway is not connected to any running API Gateway instance. The Microgateway user repository is populated by copying the API Gateway user repository (users.cnf) when provisioning a Microgateway.

The figure illustrates the Microgateway user repository being populated by copying the API Gateway user repository.



When you provision a Microgateway or start a Microgateway the users.cnf and related configurations are picked up from the location `IntegrationServer\instances\default\config\users.cnf` in the API Gateway installation directory.

The API Gateway installation directory can be specified using the Microgateway configuration parameter `apigw_dir`

The parameter can be specified either as a command line option or through the Microgateway configuration file.

The configuration parameter applies to the following Microgateway commands:

- createInstance
- createDockerFile

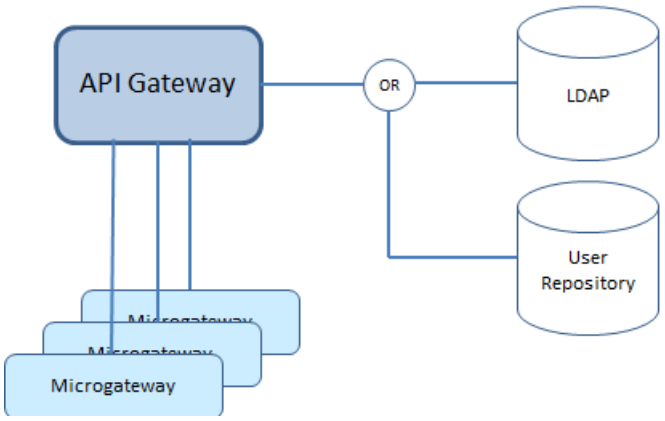
These commands create a copy of the users.cnf. The `apigw_dir` also affects the users.cnf lookup that happens during Microgateway start. The lookup procedure checks for local users.cnf (config/users.cnf). If there is no local users.cnf, then lookup users.cnf using apigw_dir. If apigw_dir or users.cnf is not found, then the startup fails.

Microgateway does not support user authentication by default. To activate user authentication you have to specify the parameter `user_auth = internal` when provisioning or starting a Microgateway.

If the user authentication is not activated, APIs with authentication policies are rejected. The users.cnf lookup is only performed when user authentication is activated.

Delegated authentication using API Gateway

Microgateway supports the delegated authentication to API Gateway. The API Gateway performs the authentication against the configured LDAP or the user repository.



The delegated authentication is activated by setting the parameter `user_auth = delagated` . When the delegated authentication is activated, Microgateway talks to the API Gateway authentication API.

The authentication API exposes the resource: /rest/apigateway/authenticate

The resource exposes a POST method. The a user authentication is triggered through the request:

```
POST /rest/apigateway/authenticate
 {
    "user-id":"",
    "password:"",
    "domain":""
}
```

The password/credentials are transferred in an unencrypted way. Therefore, the delegated authentication must happen through HTTPS.

On successful authentication the API returns a HTTP 200 response with user information and expiry information as follows:

```
HTTP 200 OK
 {
    "status": "Authenticated",
    "user": {
    },
    "expires":""
}
```

The response provides information about the user and the expiry interval. This tells the Microgateway for how long the delegated authentication result can be cached.

If the authentication fails a HTTP 401 response is returned.

Application Synchronization to support Identity and Access Management Policy

For Microgateway to support the Identity and Access Management (IAM) policies it is necessary that Microgateway has the recently updated applications from the API Gateway instance from where the applications were provisioned. Microgateway provides a mechanism to synchronize applications between API Gateway and Microgateway to support the IAM policy.

During API provisioning the applications are pulled from the API Gateway instance into Microgateway. After provisioning these applications in Microgateway, these applications have to be in synchronization with those in the API Gateway from where they were provisioned so that any changes in the applications in the API Gateway instance is reflected in the Microgateway. This helps the IAM policy for an API in Microgateway executes with the latest applications instead of verifying against the stale application data.

Application synchronization in Microgateway is achieved through a polling mechanism. To avoid the consumption of a considerable amount of memory and CPU, the API Provider provides certain configurations for polling the applications to minimize the memory and CPU utilization. The polling can be done for the below parameters:

- List of application ids
- All registered applications of the APIs in Microgateway
- All global applications

A property `applicationstoSync` is added where the user specifies `registeredapplication`, `all`, or comma separated ids.

Some considerations during the application synchronization:

- Microgateway is provisioned with this configuration before start up.
- Only one thread runs for synchronization.
- On crash of thread execution, it starts again.
- A timestamp of the last synchronized application is maintained in the Microgateway so that the next polling would be for `applications updated > timestamp`

- A property to specify polling interval is added.
- A property to enable or disable synchronization is added.

Routing

The policies in this stage enforce routing of requests to target APIs based on the rules you can define to route the requests and manage their respective redirections according to the initial request path. The policy included in this stage is Straight-through Routing.

Straight Through Routing

When you select the Straight Through routing protocol, the API routes the requests directly to the native service endpoint you specify. If your entry protocol is HTTP or HTTPS, you can select the Straight Through routing policy.

The table lists the properties that you can specify for this policy:

Parameter	Value
Endpoint URI	Specifies the URI of the native API endpoint to route the request to in case all routing rules evaluate to False. Service registries that have been added to the Microgateway instance are also included in the list. If you choose a service registry, Microgateway sends a request to the service registry to discover the IP address and port at which the native service is running. Microgateway replaces the service registry alias in the **Endpoint URI** with the IP address and port returned by the service registry. For example, if your service is hosted at the URL: `http://host:port/abc/`, you need to configure the Endpoint URI as: `http://${ServiceRegistryName}/abc/`.
HTTP Method	Specifies the available routing methods: **GET**, **POST**, **PUT**, **DELETE**, and **CUSTOM** (default). When **CUSTOM** is selected, the HTTP method in the incoming request is sent to the native service. When other methods are selected, the selected method is used in the request sent to the native service.
HTTP Connection Timeout (seconds)	Specifies the time interval (in seconds) after which a connection attempt times out. If a value 0 is specified (or if the value is not specified), Microgateway uses the default value 30 seconds.

Parameter	Value
Read Timeout (seconds)	Specifies the time interval (in seconds) after which a socket read attempt times out. If a value 0 is specified (or if the value is not specified), Microgateway uses the default value 30 seconds.
SSL configuration - Specifies values to enable SSL client authentication that Microgateway uses to authenticate incoming requests for the native API.	
Keystore Alias	Specifies the keystore alias that is present in the system-settings.yml. This value (along with the value of Client Certificate Alias) is used for performing SSL client authentication.
Key Alias	Specifies the alias for the private key, which must be stored in the keystore specified by the keystore alias.

Traffic Monitoring

The policy in this stage provides ways to enable logging request and responses to a specified destination. The policy included in this stage is Log Invocation.

The policies in this stage provide ways to enable logging request and responses to a specified destination and enforce limits for the number of service invocations during a specified time interval and send alerts to a specified destination when the performance conditions are violated. The policies included in this stage are:

- Log Invocation
- Throttling Traffic Optimization

Log Invocation

This policy enables logging requests or responses to API Gateway and external Elasticsearch. This action also logs other information about the requests or responses, such as the API name, operation name, the Integration Server user, a timestamp, and the response time.

The table lists the properties that you can specify for this policy:

Parameter	Description
Store Request	Logs all requests.
Store Response	Logs all responses.
Compress Payload Data	Compresses the logged payload data.
Log Generation Frequency	Specifies how frequently to log the payload.

Parameter	Description
	Provide one of the following options: ■ **Always**. Logs all requests and responses. ■ **On Failure**. Logs only the failed requests and responses. ■ **On Success**. Logs only the successful responses and requests.
Destination	Specifies the destination where to log the payload. Provide the required destinations: ■ **API Gateway** ■ **Elasticsearch**

Throttling Traffic Optimization

This policy limits the number of service invocations during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific clients in terms of resource usage, and so on.

The table lists the properties that you can specify for this policy:

Parameter	Description
Limit Configuration.	
Rule name	Specifies the name of throttling rule to be applied. For example, Total Request Count.
Operator	Specifies the operator that connects the rule to the value specified. Select one of the operators: **Greater Than**, **Less Than**, **Equals To**.
Value	Specifies the value of the request count beyond which the policy is violated.
Destination	Specifies the destination to log the alerts. Provide the required options: ■ **API Gateway** ■ **Elasticsearch**
Alert Interval	Specifies the interval of time for the limit to be reached.
Unit	Specifies the unit for the time interval in minutes, hours, days, or weeks for the alert interval.

Parameter	Description
Alert Frequency	Specifies the frequency at which the alerts are issued. Provide one of the options: ■ **Only Once**. Triggers an alert only the first time the specified condition is violated. ■ **Every Time**. Triggers an alert every time the specified condition is violated.
Alert Message	Specifies the text message to be included in the alert.
Consumer Applications	Specifies the application to which this policy applies.

Error Handling

The policy in this stage enables you to specify the error conditions, lets you determine how these error conditions are to be processed. The policy included in this stage is Conditional Error Processing.

Conditional Error Processing

Error Handling is the process of passing an exception message issued as a result of a run-time error to take any necessary actions. This policy returns a custom error message (and the native provider's service fault content) to the application when the native provider returns a service fault. You can configure conditional error processing and use variables to create custom error messages.

The table lists the properties that you can specify for this policy:

Parameter	Description
Error conditions. Specifies the error conditions and how these error conditions should be processed.	
Status Code Error Criteria	Specify the error status code. Provide a value for the **Code**.
Header Error Criteria	Provide the details of the custom HTTP header(s) included in the client requests. Provide the following information: ■ **Header Name**. Specifies the name of the HTTP header. ■ **Header Value**. Specifies the value of the HTTP header.
Payload Criteria	Provide the details of the payload criteria in the API request.

Parameter	Description

	Provide the following information in the payload identifier section:
	- **Expression type**. Specifies the type of expression, which is used for identification. You can select one the following expression type:
	  - **XPath**. Provide the following information:
	  - **Payload Expression**. Specifies the payload expression that the specified XPath expression type in the request or the response has to be converted to. For example: /name/id.
	  The response maybe a native service error or Microgateway generated error.
	  - **Namespace Prefix**. The namespace prefix of the payload expression to be validated.
	  - **Namespace URI**. The namespace URI of the payload expression to be validated.
	  Note:
	  You can include multiple namespace prefix and URI.
	  - **JSONPath**. Provide the **Payload Expression** that specifies the payload expression that the specified JSONPath expression type in the request or the response has to be converted to. For example: $.name.id.
	  The response maybe a native service error or Microgateway generated error.
	  - **Text**. Provide the **Payload Expression** that specifies the payload expression that the specified Text expression type in the request or response has to be converted to. For example: any valid regular expression.
	  The response maybe a native service error or Microgateway generated error.
	  You can add multiple payload identifiers as required.
	  Note:
	  Only one payload identifier of each type is allowed. For example, you can add a maximum of three payload identifiers, each being of a different type.
	Value: Specifies a value that has to match with the value contained in the error Response.

Parameter	Description
Custom Error Variables. Specifies the error variables to be used in the custom error message.	
Payload Type	Specify the payload type. Available values are: ■ **Request**. Specifies the request payload type. ■ **Response**. Specifies the response payload type.
Name	Provide a name for the payload type.
Payload Identifier	Provide the details of the payload criteria in the API request. Provide the following information in the Payload identifier section: ■ **Expression type**. Specifies the type of expression contained in the payload request. ■ **Payload Expression**. Specifies the payload expression that the specified expression type in the request has to be converted to. ■ **Namespace Prefix**. The namespace prefix of the payload expression to be validated. ■ **Namespace URI**. The namespace URI of the payload expression to be validated. **Note:** You can add multiple namespace prefix and URI. You can add multiple payload identifiers as required.
Failure Message. Specifies the custom failure message format that Microgateway should send to the application. Specify whether the message should be in the **text**, **json**, or **xml** format.	
Send Native Provider Fault Message	Enable this parameter so that Microgateway sends the native REST failure message to the application. When you disable this parameter, the failure message is ignored when a fault is returned by the native API provider.

5 Publishing APIs to Runtime Service Registries

■ Publishing APIs to Runtime Service Registries 42

Publishing APIs to Runtime Service Registries

Microgateway provides capability to publish APIs to runtime service registries.

In a micro-service landscape service registries provide the information about service instances and their location or endpoints. This information enables the service discovery during runtime. Accordingly you can configure a Microgateway instance to register all its provisioned APIs to a service registry. During startup it generates one service registry entry per API. The endpoints of the registered APIs are based on the host and port of the Microgateway instance. Multiple Microgateway instances can expose the same APIs and register them to the same service registry. The service registry then shows all the endpoints of the given APIs. During shutdown a Microgateway instance removes the service registry entries it has generated.

Microgateway uses service registries in the following way:

- You can publish the provisioned APIs to service registries. This enables other applications to use the service registry to dynamically locate a Microgateway instance that provides that API.

Microgateway currently supports the following service registries.

- **Eureka**

  Eureka is a REST-based service for locating services for the purpose of load balancing and failover of middle-tier servers. It has been primarily designed for applications in the AWS cloud.

- **Service Consul**

  Service Consul is a tool for discovering and configuring services in IT infrastructure.

You can configure a Runtime service registry in Microgateway configuration. Use the config.yml file loaded with the --config <arg> (-c <arg> as shortcut) option.

A sample config.yml file is as follows:

```
#service registry
serviceRegestrys:
   - #this symbols a list entry in yaml language, but must always be there,
even if you only use one registry
      name: "Test"
      serviceRegistryType: "SERVICE_CONSUL"
      connectionTimeout: 30
      readTimeout: 30
      endpoint: "http://localhost:8500/v1"
      coustomHeaders:
          "X-Consul-Token" : ""
      discoveryInfo:
          path: "/catalog/service/{serviceName}"
          httpMethod: "GET"
      registrationInfo:
          path: "/agent/service/register"
          httpMethod: "PUT"
      deRegistationInfo:
          path:  "/agent/service/deregister/{serviceId}"
          httpMethod: "PUT"
```

```
-
   name: "EurekaTest"
   serviceRegistryType: "EUREKA"
   connectionTimeout: 30
   readTimeout: 30
   endpoint: "http://localhost:9091"
   discoveryInfo:
       path: "/eureka/apps/{app}"
       httpMethod: "GET"
   registrationInfo:
       path: "/eureka/apps/{app}"
       httpMethod: "POST"
   deRegistationInfo:
       path:  "/eureka/apps/{app}/{instanceId}"
       httpMethod: "DELETE"
```

The table lists the variables in the config file, their description and their usage.

Variable	Description
name	Name of the registry
serviceRegistryType	Type of the registry. It is eitherEUREKA or SERVICE_CONSUL
connectionTimeout	Specifies the time, in seconds, for which the microgateway tries to connect to the registry.
readTimeout	Specifies the time, in seconds, for which the registry tries to connect to the service endpoint.
endpoint	The path of the registry.
discoveryInfo	Information required to discover the registry path. The rest path of the registry to register services http Method. The rest path of the registry to register services.
registrationInfo	Information required to register services. path. The rest path of the registry to register services. http Method. The rest path of the registry to register services.
deRegistationInfo	Information required to deregister services. path. The rest path of the registry to deregister services. http Method. The rest path of the registry to deregister services.
customHeaders	A map of all custom Headers you need to reach your registry. This is required only if you use consul with "X-Consul-Token" : "" as entry.

Variable	Description
username	*Optional.* The username of the user authorized to to register services at your registry.
password	*Required if you are using the username.* Specifies the password of the user authorized to to register services at your registry.

6 Command Line Reference

■ Microgateway Command Line Reference 46

Microgateway Command Line Reference

This section describes operations you can perform such as, start and stop Microgateway, retrieve Microgateway status, view the assets provisioned, create a Microgateway instance, create an asset archive, create a docker file, and so on through Command Line Interface(CLI).

Using Microgateway CLI

The Microgateway CLI script comes in 2 flavors: Windows (.bat) and Linux (.sh). Invoking the script provides usage information:

```
>microgateway.bat
Missing valid action
usage: microgateway <action> <options>

action:

   start                - Start a Microgateway server
   stop                 - Stop a Microgateway server
   status               - Retrieve the Microgateway server status
   assets               - Show the provisioned assets of a running server
   createInstance       - Create a Microgateway instance
   createAssetArchive   - Create an asset archive
   createDockerFile     - Create a Microgateway docker file
```

Starting a Microgateway

Run the following command to start a Microgateway.

```
microgateway.bat start <options>
```

where the options are:

-Shortcut, --Name	Description
-a, --archive <arg>	List of API Gateway archives
-apis, --apis <arg>	List of API identifiers (name, unique identifier, name/version).
-apps, --applications <arg>	List of global applications (name, unique identifier)
-c,--config <arg>	Configuration (YAML) file
-gw,--api_gateway <arg>	API Gateway URL
-gwp,--api_gateway_password <arg>	API Gateway password
-gwu,--api_gateway_user <arg>	API Gateway user
-gwd,--api_gateway_dir <arg>	API Gateway install directory

-Shortcut, --Name	Description
-lv,--log_level <arg>	ERROR, WARN, INFO, DEBUG, TRACE The default value is ERROR
-lp,--log_path <arg>	Path to log files The default value is logs
-p,--http_port <arg>	HTTP port number
-pols,--policies <arg>	List of global policy identifiers (name, unique identifier).
-sp,--https_port <arg>	HTTPS port number
-adu,--admin_user <arg>	User for administration access
-adp,--admin_password <arg>	Password for administration access
-v,--verbose	Print more information to console You will see a status message for each provisioned API. Also, the user authentication status is exposed.
-ua,--user_auth <arg>	User authentication method (internal or delegated)
-as, --apps_sync	Enable Applications sync
-asv, --apps_to_sync	Applications to sync (all, registeredApplications, comma separated ids)
-asi, --apps_sync_interval	Polling interval in secs for applications sync
-ast, --apps_sync_timeout	Connection timeout in secs for applications sync
-rs,--retain_settings	Retain settings given in system-settings.yml (true/false) The default value is false

Stopping a Microgateway

Run the following command to stop a Microgateway.

```
microgateway.bat stop <options>
```

where the options are:

-Shortcut, --Name	Description
-c,--config <arg>	Configuration (YAML) file
-p,--http_port <arg>	HTTP port number

-Shortcut, --Name	Description
-sp,--https_port <arg>	HTTPS port number

Retrieving Microgateway Status

Run the following command to retrieve the status of a Microgateway.

```
microgateway.bat status <options>
```

where the options are:

-Shortcut, --Name	Description
-p,--http_port <arg>	HTTP port number
-sp,--https_port <arg>	HTTPS port number

Viewing the Provisioned Assets in Microgateway

Run the following command to view the assets provisioned in Microgateway.

```
microgateway.bat assets <options>
```

where the options are:

-Shortcut, --Name	Description
-c,--config <arg>	Configuration (YAML) file
-p,--http_port <arg>	HTTP port number
-sp,--https_port <arg>	HTTPS port number
-v,--verbose	Print all details

Creating a Microgateway Instance

Run the following command to create a Microgateway instance package.

```
microgateway.bat createInstance <options>
```

where the options are:

-Shortcut, --Name	Description
-gwd,--api_gateway_dir <arg>	API Gateway install directory for taking over the user credential file
-c,--config <arg>	Configuration (YAML) file which will be copied into the instance

-Shortcut, --Name	Description
-os,--os <arg>	Operating system (win / linux)
-ins,--instance <arg>	Zip filename to hold the resulting Microgateway instance (mandatory)

Creating an Asset Archive

Run the following command to create an asset archive from a running API Gateway instance.

```
microgateway.bat createAssetArchive <options>
```

where the options are:

-Shortcut, --Name	Description
-a, --archive <arg>	The resulting API Gateway archive
-apis, --apis <arg>	List of API identifiers (name, unique identifier, name/version).
-apps, --applications <arg>	List of global applications (name, unique identifier, name/version).
-gw,--api_gateway <arg>	API Gateway URL
-gwp,--api_gateway_password <arg>	API Gateway password
-gwu,--api_gateway_user <arg>	API Gateway user
-pols,--policies <arg>	List of global policy identifiers (name, unique identifier).

Creating a Microgateway Docker File

Run the following command to create a Microgateway docker file.

```
microgateway.bat createDockerFile <options>
```

where the options are:

-Shortcut, --Name	Description
-a, --archive <arg>	List of API Gateway archives
-c,--config <arg>	Configuration (YAML) file
-dod,--docker_dir	Microgateway directory to use in Docker file
-dof,--docker_file	Filename to hold Docker file

-Shortcut, --Name	Description
-dor,--docker_from	FROM image to use in Docker file
-lv,--log_level <arg>	ERROR, WARN, INFO, DEBUG, TRACE The default value is ERROR
-p,--http_port <arg>	HTTP port number
-sp,--https_port <arg>	HTTPS port number

Config.yml

The following shows a sample config.yml file structure.

```
# Microgateway ports
ports:
   http: 5554
   https: 5553
   key_alias: ssos

# Microgateway API endpoint base path
api_endpoint:
    base_path: /gateway

# Microgateway admin API configuration
admin_api:
    admin_path: /rest/microgateway
    user: Administrator
    password: <pwd>

# API Gateway configuration
api_gateway:
   url: http://localhost:5555
   user: Administrator
   password: <pwd>
   dir: /opt/softwareag/IntegrationServer/instances/default
   retain_settings: true | false

# Asset archive configuration
archive:
   file: export.zip,export1.zip,export2.zip

# Specifying assets to download from API Gateway
downloads:
   apis: EmployeeService
   applications:
   policies:

# Specify the runtime authorization method
policies:
   user_auth: internal | delegated

# Microgateway logging configuration
logging:
   level: ERROR
```

```
   path: logs

# Application Sychronization
applications_sync:
  enabled: true | false
  applications_to_sync: "all | registeredApplications | comma separated ids"
  polling_interval_secs: 2
  connection_timeout_secs: 10
```

system-settings.yml

The following shows a sample system-settings.yml file structure.

```
---
faults:
   default_error_message: "API Gateway encountered an error.
   Error Message: $ERROR_MESSAGE. Request Details: Service - $SERVICE,
   Operation - $OPERATION, Invocation Time:$TIME, Date:$DATE,
   Client IP - $CLIENT_IP, User - $USER and Application:$CONSUMER_APPLICATION"
   native_provider_fault: "false"
extended_settings:
   defaultEncoding: "UTF-8"
   apiKeyHeader: "x-Gateway-APIKey"
   apig_MENConfiguration_tickInterval: "60"
   events.collectionQueue.size: "10000"
   events.collectionPool.minThreads: "1"
   events.collectionPool.maxThreads: "8"
gateway_destination:
   sendPolicyViolationEvent: "true"
es_destination:
   protocol: "http"
   hostName: "localhost"
   port: "9240"
   indexName: "gateway_default_analytics"
   userName: ""
   password: ""
   sendPolicyViolationEvent: "true"
key_store:
   type: JKS
   provider: SUN
   location: config/keystore.jks
   password: manage
system:
   version: "@full_version@"
---
```

REST APIs in Microgateway

Administration API

The Microgateway exposes a REST API for administration purpose. The methods of the API allow to query the status, the provisioned assets, and the configure settings of a running Microgateway instance.

If there are credentials configured in the Microgateway configuration, the Administration API requires a basic authentication.

GET /rest/microgateway/status

Retrieves a status message showing the version of the Microgateway.

GET /rest/microgateway/settings

Retrieves the configured settings of the Microgateway.

GET /rest/microgateway/assets

Retrieves the provisioned assets.