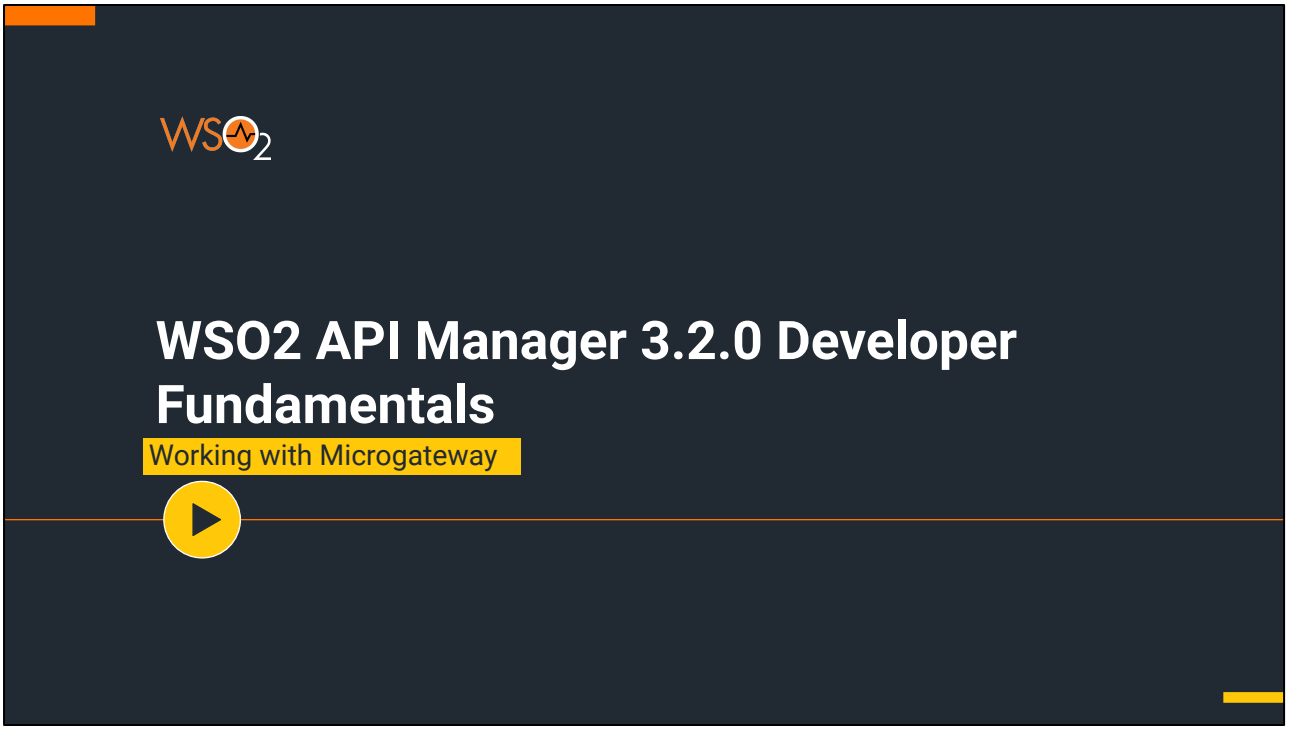

WSO2
WSO2 API Manager 3.2.0 Developer Fundamentals
Working with Microgateway

WSO2 Microgateway - Introduction

- WSO2 Microgateway is a Cloud-Native, high performance and lightweight gateway for APIs.
- It can process incoming and outgoing messages while collecting information required for usage metering and throttling capabilities.
- Microgateway is immutable*, and fits well with Microservice architecture.
- Has a very low memory footprint
- Microgateway supports automated CI/CD with CLI tools.

Immutable - Once a Microgateway is built and deployed, it cannot be modified. If any modification is required, it should be rebuilt.

Documentation: Microgateway Overview

- Designed to scale
  - Self validating tokens
  - Localize rate limiting
  - Offline analytics
  - Immutable
  - Stateless
- Native support for Docker and K8s
- Ideal to be deployed in lockdown

n summary, the API Microgateway is a simplified lightweight version of the API Gateway. However, the API Microgateway only supports a subset of the capabilities of the API Gateway

Features

Features

- Authentication : Supports mutual TLS, API Key, OAuth2(opaque tokens and JWT) and basic authentication
- Rate limiting : Rate limit the request to the APIs based on the request count for a given time period
- Transformations : Request and response message transformation and mediation using interceptors
- Analytics : Publish data to streams
- Service Discovery : Resolve endpoints using third party distributed key value stores like etcd

Features Contd...

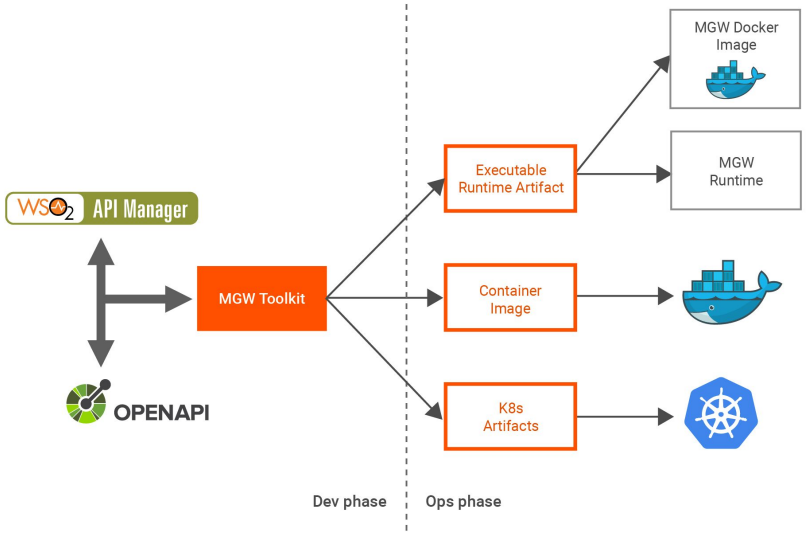
- Cloud native : A lightweight gateway that can be run on any platform(bare metal, docker and k8s)
- Scalable : Distributed nature allows to scale horizontally.
- Private Jet mode : Enable to deploy APIs in a dedicated Gateway node in Kubernetes

The API Microgateway natively supports scaling in highly decentralized environments including microservice architecture.

The API Microgateway is also capable of operating in lockdown environments such as IoT devices since connectivity from the API Microgateway to the API Management system is not mandatory.

Microgateway Architecture

Architecture



- API developer creates a WSO2 API Microgateway project using a WSO2 API Microgateway controller(toolkit)
- Adds the open API definitions of microservices into the project
- Developer defines endpoints and interceptors for the api/resources using the definition.yaml inside the project
- Builds the project and generates executables, images and k8s artifacts

Toolkit

- Command line interface (CLI) to manage microgateway projects
- Initialize microgateway projects with open API definitions
- Builds the projects to create
  - Runtime artifacts for microgateway runtime and docker runtimes
  - Immutable containers with APIs built in
  - Kubernetes artifacts used to deploy in k8s clusters
- Can import APIs from WSO2 API Manager

Runtime

- Serves the requests applying
  - Security
  - Rate limiting
  - Transformations
  - Analytics and etc
- Available as archived distributions as well as docker images[1]
- APIs can be burned into container images to spawn immutable containers
- Runs using artifacts generated by the toolkit

[1]- https://hub.docker.com/r/wso2/wso2micro-gw

Deploying API in Microgateway

Initiating a Microgateway Project using Toolkit

A Microgateway project can be initialized as follows.

1. As an empty project
   - `micro-gw init petstore`
2. Using an existing API definition
   - Initialize a project with Microgateway Toolkit using an API Definition.
     `micro-gw init petstore -a https://petstore.swagger.io/v2/swagger.json`
   - This generates a project directory with the required artifacts.

Microgateway Project Structure

```
petstore

├── api_definitions
    └── swagger.json
├── conf
│   └── deployment-config.toml
├── extensions
│   ├── extension_filter.bal
│   ├── startup_extension.bal
│   └── token_revocation_extension.bal
├── grpc_definitions
├── interceptors
├── lib
├── policies.yaml
```

Detailed information on Microgateway Project structure:
https://mg.docs.wso2.com/en/3.2.0/reference/project-directory/

https://mg.docs.wso2.com/en/latest/reference/project-directory/

Build and Run the Project

- Build the Microgateway project using the toolkit.

  ```
  micro-gw build petstore
  ```

  - This command will generate the microgateway executable file in the target directory of the project. (petstore.jar)
- To run the API in Microgateway,
  - Go to the Microgateway extracted location/bin directory.
  - Execute

    ```
    gateway <MICRO_GW_PROJECT_PATH>/target/petstore.jar
    ```
  - The Microgateway will start on port 9095 and the api can be invoked as follows.

    ```
    curl -X GET "https://localhost:9095/v2/pet/1" -H "accept:
    application/xml" -H "api_key:$TOKEN" -k
    ```


Please refer the Quick Start Guide for more details.

Securing APIs in Microgateway

API Security in Microgateway

- WSO2 Microgateway supports the following API Authentication mechanisms.
  - OAuth2 (JWT/ Opaque token)
  - Basic Authentication
  - API Key
  - MutualSSL (Transport Security)
- Microgateway is capable of validating self contained JWT type tokens.
- If Opaque tokens (reference tokens) are used, an external key manager must be configured.
- When configured with WSO2 API Manager as Key Manager, it is possible to do scope and subscription validation for APIs.
- API Security should be configured in API Definition file in the Security Scheme object.

- Security schemes can be enforced per API or per resource basis.
- If a security scheme is not specified or security is not disabled manually, all the APIs deployed in Microgateway are secured with OAuth2 by default.

Defining Security Schemes

```
components:
  securitySchemes:
    ApiKeyAuth:        # arbitrary name for the security scheme
      type: apiKey
      in: header       # can be "header" or "query"
      name: X-API-KEY  # name of the header, query parameter
    appId:             # you can define several apikey security schemas
      type: apiKey
      in: header
      name: X-APP-ID
```

Applying Globally (API Level)

```
security:
  - ApiKeyAuth: []
```

Applying Security Schemes Resource Level

```
"/pet/{petId}":
    get:
      tags:
      - pet
      summary: Find pet by ID
      description: Returns a single pet
      operationId: getPetById
      parameters:
      - name: petId
        in: path
        description: ID of pet to return
        required: true
        schema:
          type: integer
          format: int64
      security:
        - ApiKeyAuth: []
```

API Authentication

API Key Generation

- WSO2 Microgateway is capable of generating API Keys with integrated Security Token Service (STS).
- API Keys generated will be self contained JWT type tokens.
- STS configuration can be done through the `<MGW_RUNTIME_HOME>/conf/micro-gw.conf` file.
- The STS endpoint is secured with Basic Authentication.

  E.g.,:

  ```
  curl -k -X GET "https://localhost:9095/apikey" -H  "Authorization:Basic YWRtaW46YWRtaW4="
  ```

  API Key Security Token Service

Authorization

- Scopes are used to validating the rights to access the requested API resource.
- Microgateway provides scope validation for below API security types
  - OAuth2 tokens (JWT and Opaque)
  - Basic authentication
- OAuth2 scheme is defined under components/securitySchemes with all supported scopes.
- If multiple scopes are defined under a security scheme, they will be in OR relationship. i.e. Providing at least one scope of them, is enough to authorize the request successfully.

Subscription Validation

- Configurable for JWT and reference tokens
- Microgateway will validate the list under the subscribedAPIs claim and check if the user is currently invoking one of the APIs in the list.
- Subscriptions are validated in the gateway it self by using a set of internal data stores.
  - Application-key mapping
  - Application data
  - API data
  - Subscription data

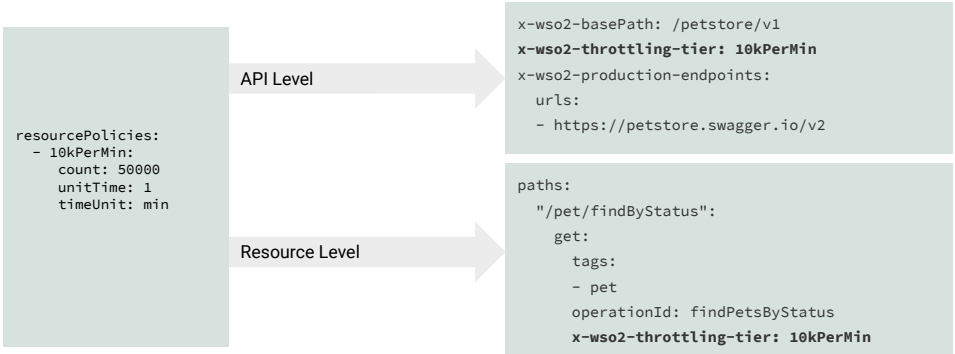
Rate Limiting

Rate Limiting

- Similar to the general gateway, Microgateway also supports enforcing throttling and rate limiting policies for APIs.
- Microgateway can throttle based on local counters (local throttling)
- Microgateway can work with Traffic Manager to cater advanced throttling scenarios. (distributed throttling)
- Microgateway supports following levels of Rate limiting.
  - Application level
  - Resource level
  - Subscription level
- Microgateway supports custom Siddhi based throttle policies with the aid of Traffic Manager of APIM.

Defining and Enforcing Policies

- Throttling policies are defined in the policies.yaml file.
- To enforce policies to APIs, add the required policy in the open api definition. (API Level or resource level)



Adding Throttling Policies

Resiliency

Resiliency

- Microgateway is capable of handling the scenarios where backend might be unavailable due to load or backend is taking lot of time to respond.
- Mechanisms used are
  - **Retry Config** - specifies how the gateway retries in the event of an error when connecting with the backend.
  - **Timeout Config** - used to gracefully handle network timeouts that occurs in connecting to upstream backend services.
  - **Circuit Breaker** - specifies how to respond to subsequent requests to the same back end. This kind of mechanism is required to prevent overloading backend services during a high load.

Retry Config - The retry parameters can be specified in the x-wso2-production-endpoint or x-wso2-sandbox-endpoint open API extension

Timeout Config - When the `*timeoutInMillis*` config is specified, the Microgateway will wait until specified time period before closing the connection with the backend and sending the error response to the client.

Circuit Breaker - Based on the config, the circuit will be remain close allowing back end to be queried, or circuit will be open where gateway will not contact the backend and respond from the gateway itself.

Microgateway Use Cases

Microgateway Use Cases

Development Phase

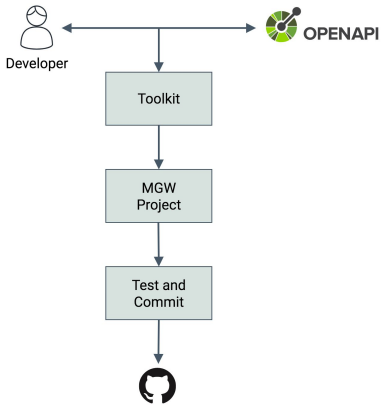
- Team of developers developing an online book store
- Members collaboratively develop services and expose those as APIs
- Each developer locally test the API and maintains the microgateway project
- Completes developer testing

Operations Phase

- Operations team deploys development environment
- Use the same build artifacts and deploys into test staging and production environments

Developer exposes 1st service, initiating microgateway project

- Developer starts creating microservice
- Defines the open API definition for the microservices
- Initiates microgateway project from open API definition
- Builds the microgateway project
- Locally tests the service exposed via microgateway
- Commits the project to the github



Developers collaboratively develop the project

- Developers collectively develop other microservices
- Check out the microgateway project
- Modify the project to add the newly added microservices
- Build the microgateway project with multiple services
- Locally test the specific microservice via gateway
- Individually each developer commits changes to the project

Operations process - Deploy in Dev

- Individual microservices are included in the microgateway project
- Request comes to deploy in the development environment
- Operations team checkouts the project
- Creates the deployment configuration for the project
- Build the project with deployment configuration
- Deploy the Microgateway using the build artifacts (In Docker/k8s)

Let’s Try it Out!

Creating and Deploying an API in Microgateway