WSO2 API Manager 3.2.0 Developer Fundamentals

Working with API Controller

WSO2 Training

Objective

At the end of this module, attendees will get a basic idea of

- What is API Controller?
- Creating an API Project with dev-first approach
- Migrating APIs and API Products to Different Environments
- Migrating Applications to Different Environments
- API Controller Functions

API Controller - Introduction

API Controller

- API Controller is a command line tool which can be used for managing APIs, API Products and Apps.
- API Controller can be used to,
  - Create and import APIs and API Products
  - Manage Environments
  - Migrating APIs and API Products among environments. (dev, staging, production etc)
  - Build CI/CD pipeline for APIs
  - Migrating Applications among environments
- Has the ability to work with kubernetes

Creating an API via Dev-first Approach

Dev-First Approach of API Creation

- Usually in API Manager, an API is created via Publisher portal.
- API CTL can be used to create an API from scratch or using an existing Swagger or Open API specification
- Dev-First approach introduces a Project based approach, which can be initialized using API CTL.
- In this approach, the project is based on an API definition file (swagger) and other artifacts such as mediation sequences etc are also can be associated with the project.
- This approach simplifies the API creation and management and helps automating the process.

Downloading and Initializing the API CTL

- Navigate to the API Management Tooling page
  https://wso2.com/api-management/tooling/
- Under **Dev-Ops Tooling**, click **Download** based on your preferred platform
- Extract the downloaded archive of the CTL Tool to the desired location
- Navigate to the working directory where the executable CTL Tool resides
- Execute the following command to start the CTL Tool

```
./apictl
```


Documentation : [Getting Started with API Controller]

Creating API Project Using API CTL

An API Project can be created as follows.

1. As an empty project from scratch

```
apictl init SampleAPI
```

2. With existing API definition file.

```
apictl init Petstore --oas petstore.yaml
```

Executing the above commands will generate an API Project with required artifacts.

Documentation : [Creating APIs in Dev first approach]

API Project Structure

```
├── api_params.yaml
├── Docs
│    └── FileContents
├── Image
├── Meta-information
│    ├── api.yaml
│    └── swagger.yaml
├── README.md
└── Sequences
    ├── fault-sequence
    ├── in-sequence
    └── out-sequence
```

Sub Directory/File	Description
`api.yaml`	The specification of the created API.
`swagger.yaml`	The Swagger file generated when the API is created.
`api_params.yaml`	Contains environment-specific details.

`Sequences`
`├── fault-sequence`
`├── in-sequence`
`└── out-sequence`
	To add custom sequences, save them in XML format and add them to the corresponding folder. For example, to add a custom in-sequence, save the custom sequence as `SampleSequence.xml` and add it to the `Sequences/in-sequence/` directory.

`Docs` `└── FileContents`	Contains the documents. To add a documentation, add them to the `Docs/FileContents/` directory.

Changing Default Values

The generated project contains pre-configured set of values. Therefore, before publishing the API, proper values should be provided for mandatory fields in api.yaml file. (apiName, context, productionUrl and sandboxUrl)

Sample configuration:

```
id:
    providerName: admin
    apiName: "SampleAPI"
    version: 1.0.0
type: HTTP
context: "/samplecontext"
availableTiers:
    - name: Unlimited
status: PUBLISHED
visibility: public
transports: http,https
productionUrl: http://localhost:8080
sandboxUrl: http://localhost:8081
```


Please refer to https://github.com/wso2/product-apim-tooling/blob/master/import-export-cli/box/resources/init/sample_api.yaml for more detailed api.yaml file.

Changing Environment Specific Parameters

- Parameters which vary from environment to environment such as production and sandbox endpoints etc can be provided via the api_params.yaml file.
- This file can be placed anywhere in the project directory and also a desired name can be used for the file name.
- --params flag can be used to specify the file if the file name is different.

```
apictl import-api -f dev/PhoneVerification_1.0.zip -e production --params
/home/user/custom_params.yaml
```

- Sample api_params.yaml file is available in the next slide.

Documentation : [Configuring Environment Specific Parameters]

```
environments:
    - name: <environment_name>
      endpoints:
          production:
              url: <production_endpoint_url>
              config:
                  retryTimeOut: <no_of_retries_before_suspension>
                  retryDelay: <retry_delay_in_ms>
                  factor: <suspension_factor>
          sandbox:
              url: <sandbox_endpoint_url>
              config:
                  retryTimeOut: <no_of_retries_before_suspension>
                  retryDelay: <retry_delay_in_ms>
                  factor: <suspension_factor>
      security:
        - enabled: <whether_security_is_enabled>
          type: <endpoint_authentication_type_basic_or_digest>
          username: <endpoint_username>
          password: <endpoint_password>
      gatewayEnvironments:
        - <gateway_environment_name>
      certs:
        - hostName: <endpoint_url>
          alias: <certificate_alias>
          path: <certificate_file_path>
```

Configuring Different Endpoint Types

- Different types of endpoints can be specified in the api_params.yaml file
- Following endpoint types are supported.
  - HTTP / Rest endpoints
    - Without load balancing or failover
    - With load balancing
    - With failover
  - HTTP / SOAP endpoints
    - Without load balancing or failover
    - With load balancing
    - With failover
  - Dynamic endpoints
  - AWS Lambda endpoints
    - Using IAM role-supplied temporary AWS credentials
    - Using stored AWS credentials

Documentation :
https://apim.docs.wso2.com/en/3.2.0/learn/api-controller/advanced-topics/configuring-different-endpoint-types/

Adding API Environments

- In order to publish the created API, a running API Manager instance should be configured as an Environment in API Controller
- It is possible to add environments by either manually editing the `<USER_HOME>/.wso2apictl/main_config.yaml` file or by running below command

```
apictl add-env -e <environment-name> \
       --registration <client-registration-endpoint> \
       --apim <API-Manager-endpoint> \
       --token <token-endpoint> \
       --admin <admin-REST-API-endpoint> \
       --publisher <Publisher-endpoint> \
       --devportal <DevPortal-endpoint>
```

Required :

`--environment` or `-e` : Name of the environment to be added AND (either)

`--apim` : API Manager endpoint for the environments OR (the following 4)

`--registration` : Registration endpoint for the environment

`--admin` : Admin endpoint for the environment

`--publisher` : Publisher endpoint for the environment

`--devportal` : DevPortal endpoint for the environment

Optional :

`--token` : Token endpoint for the environment

Adding API Environment

Example:

All In One

```
apictl add-env -e production \
            --apim https://apim.com:9443
```

Distributed

```
apictl add-env -e production \
            --registration https://idp.com:9444 \
            --token https://gw.com:8244/token \
            --admin https://apim.com:9444 \
            --publisher https://apim.com:9444 \
            --devportal https://apps.com:9444
```

Log in to the Environment

- Once the Environment is added, it is required to log in to the environment in order to deploy the API in that environment.
- Logging in to the environment can be done in one of the following ways.

```
apictl login <environment-name> -k
```

```
apictl login <environment-name> -u <username> -k
```

```
apictl login <environment-name> -u <username> -p <password> -k
```

Example

```
apictl login production -u admin -p admin
```

Importing the API to the Environment

- After login, the API can be deployed in the Environment.
- For this the import-api command is used. The lifecycle status of the API (Created/ published etc) will be based on the status which is defined in the api.yaml file.

```
apictl import-api -f <path to API Project> -e <environment>
```

Example:

```
apictl import-api -f SampleAPI -e production
```

Important: By default, only the users with **admin** role can import the APIs. To support other users, follow these instructions.

Migrating APIs and API Products to Different Environments

Migrating APIs and API Products to Different Environments

- This feature allows to export and import APIs or API Products between environments without creating them again from scratch.
- Login to the respective API Manager environment before exporting or importing APIs or API Products.
- It is possible to export a specific API/API Product or export all the APIs belonging to given tenant in either JSON or YAML formatted archives.
- Update the api_params.yaml file if environment specific configurations need to be changed.
- Exported API/API Product archive can be used to either import the API/API Product or seamlessly update an existing API/API Product.

Exporting and Importing APIs

- Export an API

```
apictl export-api -n <API-name> -v <version> -r <provider> -e <environment>
--preserveStatus=<preserve-status> --format <export-format> -k
```

- Export all the APIs of a tenant

```
apictl export-apis --environment
<environment-from-which-artifacts-should-be-exported> --format <export-format>
--preserveStatus --force -k
```

- Import an API

```
apictl import-api --file <path-to-API-archive> --environment <environment>
--preserve-provider=<preserve_provider> --update=<update_api>
--skipCleanup=<skip-cleanup> --params <environment-params-file>  -k
```

Exporting and Importing API Products

- Export an API Product

```
apictl export api-product -n <API Product-name> -r <provider> -e <environment>
--format <export-format> -k
```

- Import API Product

```
apictl import api-product -f <path-to-API-Product-archive> -e <environment> -k
```

- Import API Product with dependent APIs

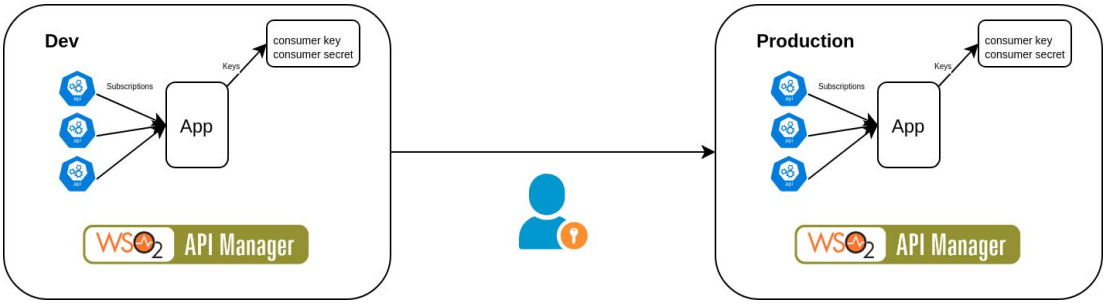
```
apictl import api-product -f <path-to-API-Product-archive> -e <environment> -k
--import-apis
```

- Import API Product to update an existing API preserving the provider

```
apictl import api-product -f <path-to-API-Product-archive> -e <environment>
--preserve-provider=<preserve_provider> --update-apis=<update_apis>
--skipCleanup=<skip-cleanup> -k
```

Migrating Applications to Different Environments

Migrating Applications to Different Environments



Exporting Applications

- Export an Application using the tool

```
apictl export-app -n MyApplication -o admin -e dev
```

  - This command exports an application named as MyApplication from Owner admin in dev environment. Application will be exported as a ZIP archive.

- Export an Application with OAuth consumer key and secrets

```
apictl export-app -n MyApplication -o admin -e dev --withKeys=true
```

  - This command exports an Application named as MyApplication from Owner admin in dev environment with the consumer key and secret as specified in --withKeys flag.

Importing Applications

- Import Applications to another environment using the tool

```
apictl import-app -f MyApplication.zip -e test --update
```

  - This will import the MyApplication.zip to the test environment. This will import all the consumer keys, subscribers into the next environment by default. Update flag will update an existing application if already exists in next environment, if not it will create a new one.

  - If application already associated with consumer key/secret in the new environment for this application, existing keys in new environment will be not be impacted.

- User can disable importing keys or subscriptions

```
apictl import-app -f MyApplication.zip -e test --skipKeys=true
--skipSubscriptions=true
```

  - Note: By default the applications will be imported under current user executing the command, User can preserve the original user using --preserveOwner flag or change the ownership to a desired owner using --owner flag

API Controller Functions

API Controller - Functions Reference

API Controller can also perform these functionalities.

- Environment:
  - Add Environment
  - List Environments
  - Remove Environments
  - Login and Logout from Environments
- APIs and API products
  - Initiating API Projects
  - List APIs / API products in an Environment
  - Filter API listing by providing a query
  - Export API / API product from an Environment
  - Import an API / API Product to an Environment
  - Change the Lifecycle State of an API
  - Delete API / API product from an Environment
- Apps
  - List Applications in an environment
  - Delete Application from an Environment
  - Export Application from an Environment
  - Import Application to an Environment
- Mode
  - Setting working mode. Available options, Default, Kubernetes

Let's Try it Out

Creating an API Project Using API CTL