Contract Testing Best Practices

# Overview

Contract testing is the validation of a deployed or mocked services endpoint to make sure the service adhere to the published contract. Note: it is recomedned to follow some open standards guide lines to pubslish contract e.g. Open API 3 (https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md), Swagger (open API 2) (<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>) , or API blue print (https://apiblueprint.org/). In general contract testing validates external service calls, and since the format of the data matters more than the actual data, it is not always necessary that the service returnyou the real data, but testing against a deployed service would be a recommended way to ensure that services (such as an API provider and a client) can still communicate with each other after development changes have been made. Interfaces that cannot be used directly because they are "production"endpoints can be tested using mock or test double (<https://martinfowler.com/bliki/ContractTest.html>).

# Some terminologies related to Contract Testing

## Service Consumer

* A component that initiates a HTTP request to another component, e.g. a Web UI client, other services and applications

## Service Provider

* A server that responds to an HTTP request from another components; one example of service provider could be services deployed to APIgateway

## Service advertisement

* The method through which a service provider published a rest based service by embedding or rendering machine understandable service descriptions e.g. a Swagger or Open API as a reference document

## Contract file

* The JSON, YAML or other file format that can be serialized and later used in API interaction or validation process (e.g. open API specification, YAML or JSON file etc.). Note: In order for contract testing to take place on a deployed service or via mocks, a contract must be available to all parties

# A good contract test should validate at least

* The API under test understands the request as published by the contract
* The API under test can send the response that is expected to understand by the client as published by the contract
* All the resources and the entities adhere to schema published by the contract
* The contract must be tested automatically (e.g. via CI/CD pipeline)before the service deployed to production e.g. in QA or Staging

# A contract test should not try to test any thing out side the contract e.g.

* Customer based workflows
* Some UI functions
* Service level agreements (SLAs)
* Failover plans
* Performance under stress
* Suitability of deployed environments

# Tools

* There are many tools that can be utilzed for contract testing, but we will focus on Postman and a command runner from postman called newman
* Pact: https://github.com/realestate-com-au/pact
* Postman: https://www.getpostman.com/
* Neman: https://github.com/postmanlabs/newman
* Dredd: https://dredd.org/en/latest/quickstart.html#install-dredd
* Hikaku: <https://github.com/codecentric/hikaku>

# Contract testing best practices for developers

* Developers are responsible for creating and exporting postman collection for any new service they are going to write
* Exported file should live some where in the code base application folder e.g. “./contract testing/[exported collections]”
* Make sure the exported collection is executed and tested via newman before you checked in the code to git repository
* The newman can be run as one of maven default lifecycles e.g. build, but constraining your development build with newman task can slow down your development lifecycle, and is not recommended
* Rather it is recommended to use maven exec plugin to run the newman. Running exec plugin gives you flexibility of not constraining your development build with newman task, and later it helps integrating newman to CI/CD build pipeline by calling mvn exec:exec from the CI/CD pipeline scripts

# Writing contract test using postman (best practices)

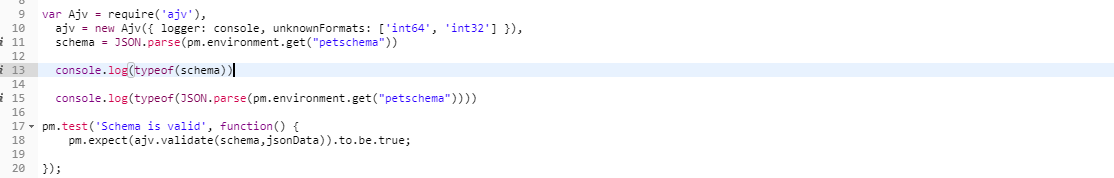
Postman test scripts are based on chai.js. For a complete list of libraries supported by Postman sandbox and Postman sandbox API reference please refer the API reference link

* Always test the structure of your response against the API contract schema. You can use the ajv framework (part of postman scripting framework) for validating the JSON response

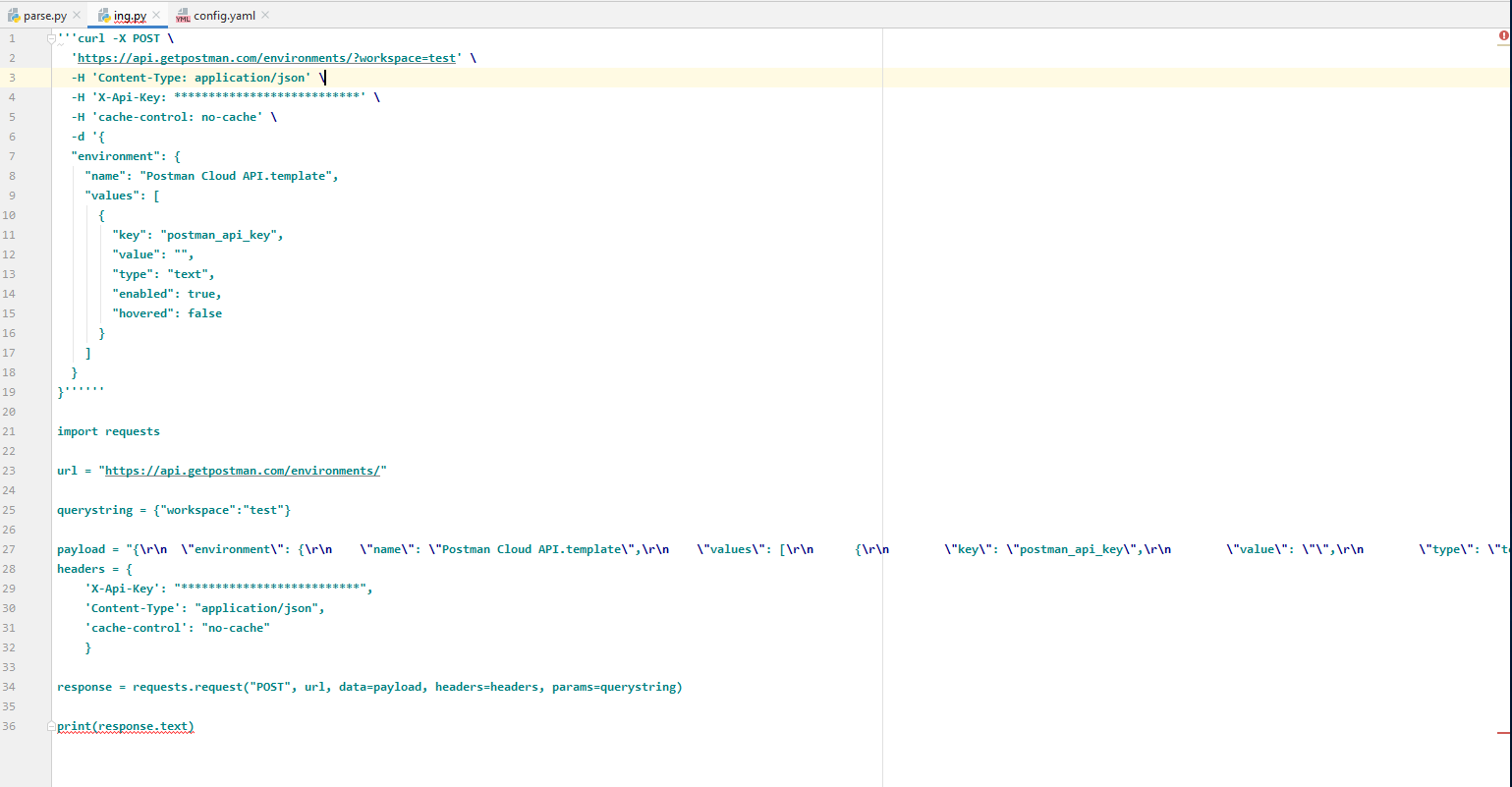
Note: the use of Tiny validator has been deprecated from postman, and it is now recomended to use ajv json schema validator. Any one who uses ajv to validate the json response for contract testing, should know that OpenAPI 3.0 uses an extended subset of JSON Schema Specification Wright Draft 00 (aka Draft 5) to describe the data formats. "Extended subset" means that some keywords are supported and some are not, some keywords have slightly different usage than in JSON Schema, and additional keywords are introduced. For the supported and unsupported json schema keywords please visit swagger open API 3.0 Supported JSON Schema Keywords section. Note: Use of $ref is supported by ajv, see the example below how to use ajv in postman



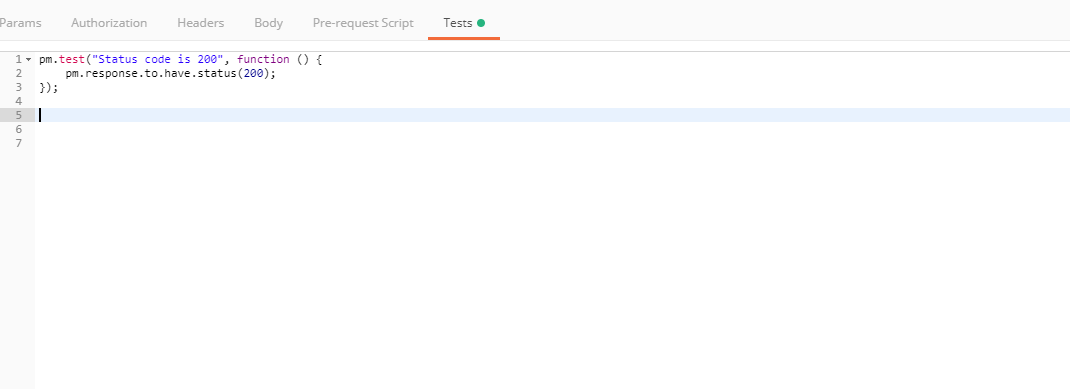
* Use environment variable to save the reference to schema and use environment variable to read your schema in your test script. Note: reading environment variable will return the json object as string, and you need to transform the javascript string as json object, see the code below for an example



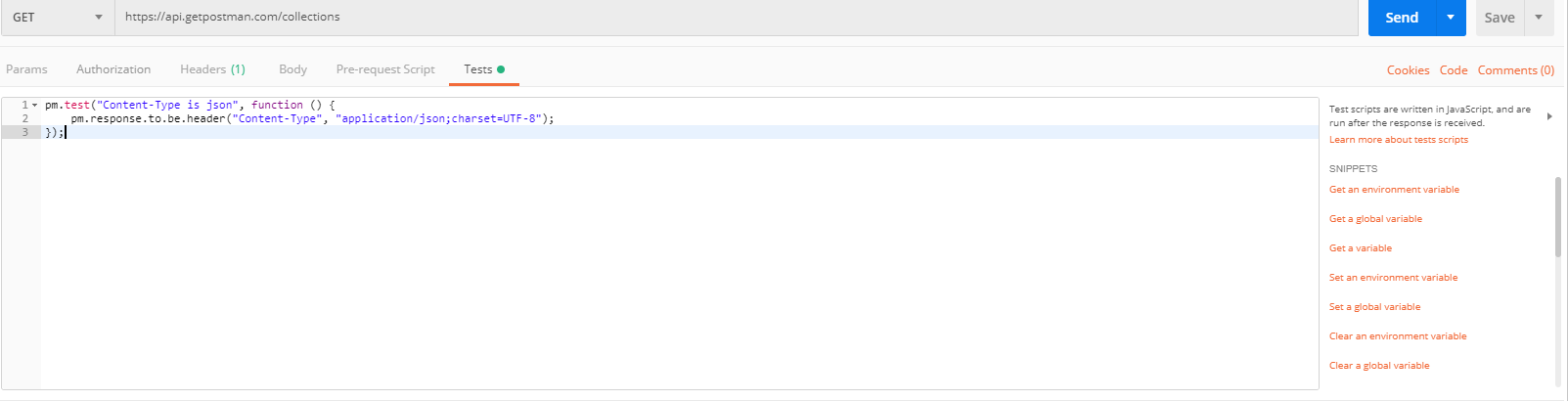
* Use postman API to create, parse and add the schema to environment variable



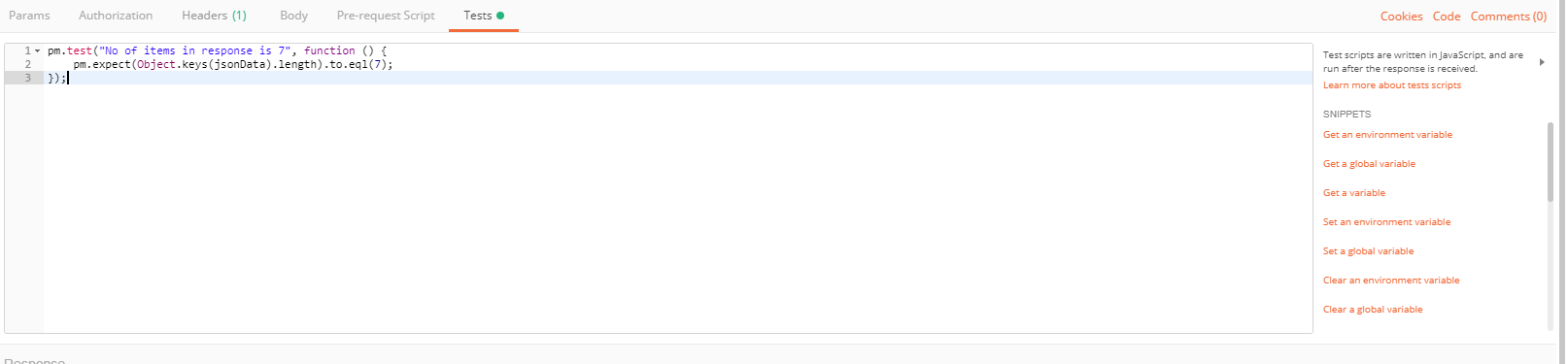
* Check all the status codes that is part of the service contract



* Check the response headers for all the important fields, e.g. version (if version is part of response header), accepted data format (JSON, XML)etc.



* Check the expected length of object or number of elements part of the JSON response



# Managing Collections

Postman provides features that helps version control many artifacts including postman collections. Collection versioning helps Synchronization, Short- and long-term undo, Branch and merge, as wells as Tagging and versioning. Note: in order to use the version control for collection, postman requires the use of professional version of the Postman called Postman Pro (<https://www.postman.com/pricing/>) . For a quick review about the difference Postman versions and the plans and pricing they support, please visit the following plan and pricing link. it is highly recommended to use the version control for collections to make sure the contract testing and the related collections are versioned and tagged with the same key word as the API code base. E.g. if API code base has a tag with version 1.1, the collections should use the same tag to reflect the contact testing for the corresponding release. For a quick review about how to version control the postman collections please visit the Version Control for Collections (<https://learning.postman.com/docs/collaborating-in-postman/version-control-for-collections/>) tutorial.

# Example project, open API JSON contract file, exported collections and environment file

In order to test the above best practices and recommendations you can import the following collection and environment files in postman, and then run the

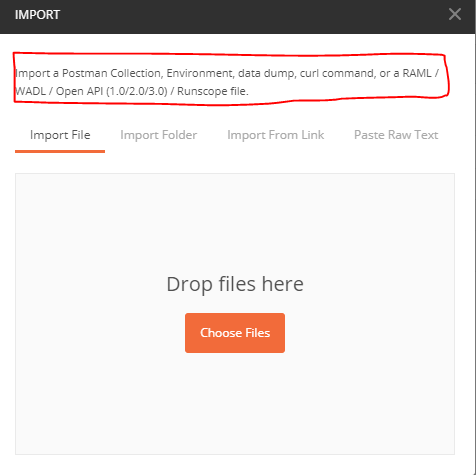
Spring Music application from the command line. You can also use the attached open API spec file to create the collection and your environment from scratch

# How to run Spring Music project from command line

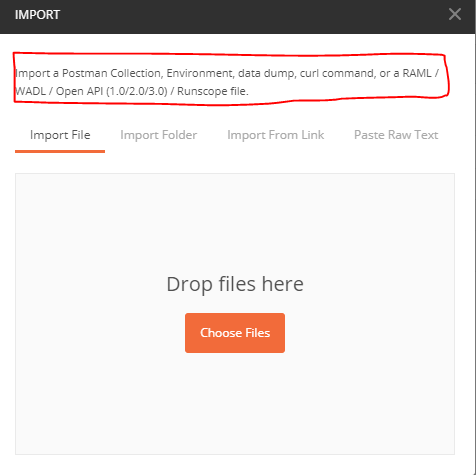
* mvn clean package
* java -jar target/springmusicexample-1.0.jar
* Application will be up and running at http://localhost:8080
* Open API JSON contract will be available at http://localhost:8080/v3/api-docs
* Open API UI and HTML document will be available at <http://localhost:8080/webjars/swagger-ui/index.html?url=/v3/api-docs&validatorUrl>=

# Best Practices Using Postman and newman for Contract Testing

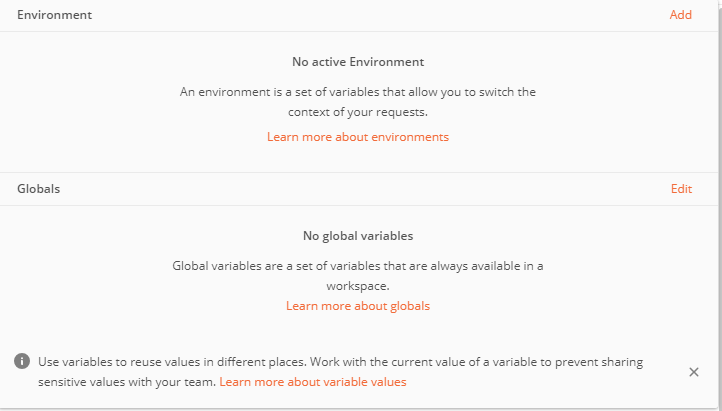
* Make sure you start your tests by importing the open API or swagger contract file, it is expected that the contract file in format of JSON or YAML file is already available at this point



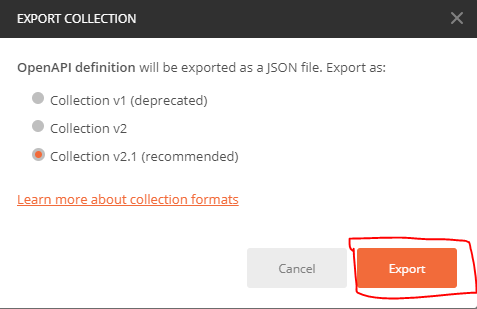
* Once you import the file the postman will automatically create a collection for you



* Use separate postman environment and global variables to store all the variables required by the rest calls. Note: use of Postman global variable is not recommended and it always better to use environment variables for storing all the variables at this point. These variables could be the server URL, API keys, Oauth secrets, query string, and path id parameters. Not embedding these variables in your collection will promote better flexibility, security, and separation of concerns that is considered as good practice. Later you can configure different environment files for different environments (E.g. QA, Staging, Production), that can be passed to newman command line based on the environment where the script is running



* Export collections to your application folder, Note you can store and save collections to Postman SAAS offering too, but since this exported collection will be used from the CI/CD pipeline is it not recommended to have an external dependency on any external API calls. Having a third party API dependencies can slow your CI/CD scripts and can introduced one extra point of failure that is considered as an anti pattern



# Running newman from the command line

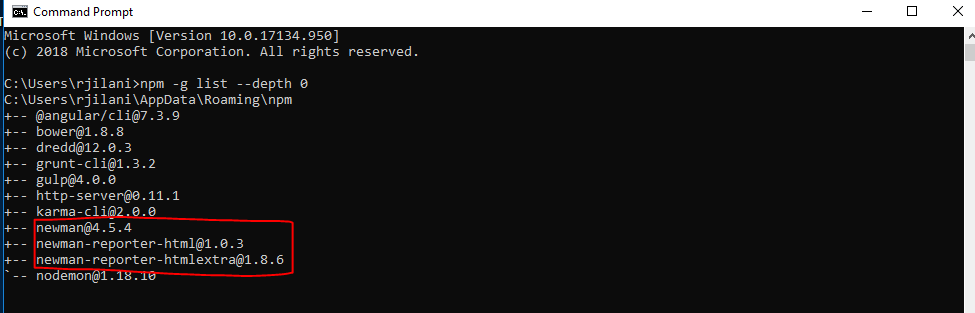
Make sure newman, and other reporting tools used by newman is installed on your machine, use tutorial below to see how to install newman and other reporting dependencies

https://www.npmjs.com/package/newman

https://github.com/DannyDainton/newman-reporter-htmlextra

https://github.com/postmanlabs/newman-reporter-html (simple report plugin, you may not need it if you have already installed new newman-reporterhtmlextra plugin above

Before running the newman commands below, run the command npm -g list --depth 0 to make sure all the required software have installed correctly

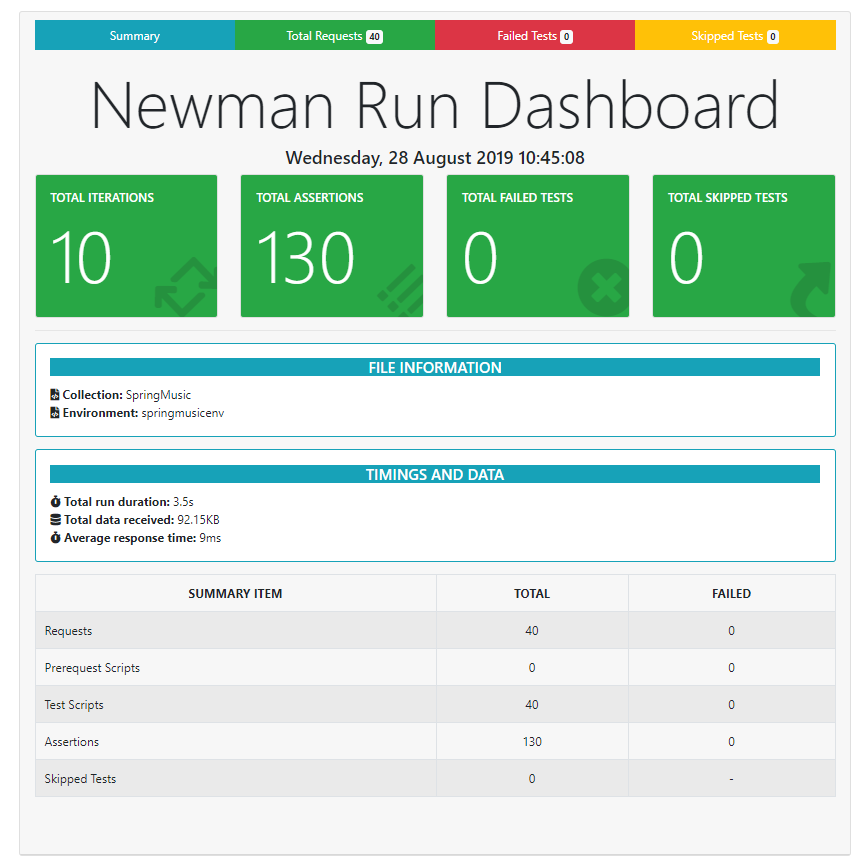


# Examples of running newman command line

* newman run -n 10 --delay-request 10 -r html [exported collections] (n is number of iterations, and d is delay in ms)
* newman run -n 10 --delay-request 10 -e [exported env] [exported collections] (use -e to pass environment variable file to newman)
* newman run -n 10 --delay-request 10 -e [exported env] [exported collections]-r htmlextra (generating html extra report)
* newman run -n 10 --delay-request 10 -[exported env] [exported collections] -r htmlextra --reporter-htmlextra-logs (generating htmlextra report with console logs) Note: running console log is only recommended when you are debugging your newman script

For all reporting options for newman htmlexta plugin please refer this link (<https://github.com/DannyDainton/newman-reporter-htmlextra>)

A demo report generated using newman-reporter-htmlextra



# Integrating newman with Jenkins

Please use the link below to see how you can integrate newman with jenkins

<https://learning.getpostman.com/docs/postman/collection_runs/integration_with_jenkins/>

Note: Integrating newman with CI/CD pipeline is subject to the tools (e.g. Codebuild, Codepipeline, Codedeploy) used by pipeline chain

# How to configure maven exec plugin

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>${maven.exec.version}</version>

<configuration>

<executable>runnewman.bat</executable>

</configuration>

</plugin>

**You can run the maven exec plugin from command line using**

**mvn exec:exec**