Efficient and Platform Independent CLI Tool for API Migration

Anuradha Jayakody[1], A.K.A. Perera[2], G. L. A. K. N. Perera[3], V. P. Wijayaweera[4], M. A. M. Asbar [5]
Sri Lanka Institute of Information Technology,Malabe ,Sri Lanka.
[1]anuradha.j@sliit.lk, [2]akayeshmantha@gmail.com, [3]kaveeperera@gmail.com,
4prabhavi313@gmail.com,[5]asbar.crate@gmail.com

***Abstract*—Development organizations maintain separate environments for development, quality assurance and production etc. These environments execute independently and have their own deployment and own methods of traffic controlling that are handled locally. In such a process artefact allowed to be created only at development environment, tested in QA environment and then would promote to the production environment for promotion. In a API managing product company when moving API management products from one environment to another, all the created APIs need to migrate across different environments to save the developer time and effort at various environments. In order to achieve accurate and efficient migration, artefacts should be portable and transferable to any new environment without any major post migration changes and additional effort.**

In this paper, we present a more powerful, efficient and generalized CLI tool that can be used by any REST based API managing applications to perform API migration in a more precise manner. We analyzed the current migration techniques use by trending API Management products and identified the major sieve point that needed to be addressed. Taking those faults n to consideration we identified the appropriate mechanism to migrate APIs across different environments.

Keywords—API migration, REST API, CLI tool, platform independence

I. INTRODUCTION

API management is the process managing activities related to API creating, publishing, monitoring and life cycle management of the APIs. There are several commercially available as well as open source products standardize this management activity. Before such a product been released in-to the market, it traverses through different environments in the organization as Dev, QA to verify the product is ready to release. While the product been proceeding at various environments, all the APIs created in one environment should be passed in-to another to examine the expected functionality uniquely across several environments. This can save the developer's time in recreating APIs in new environment as well.

Migration of API comprises of two key processors as exporting and importing. Exporting refers to moving of APIs from one environment to another and import refers to receiving APIs in-to a new destination environment. Export of APIs involves in retrieving all the API resources including API definitions, swagger definitions, thumbnails, WSDLs and documentations to another environment. During the process of API import, should be able to rebuild the API using the received API resources.

This paper organized around presenting our product, API migration tool which can perform API migration on behalf of the excessive manual work. Remainder of the paper organized as follows: the next section will point out the drawbacks we identified in current existing migration mechanisms. Then we will discuss about our product and its functionality. In the following section, we will list our findings. Finally, we conclude with the contribution of this work.

II. BACKGROUND

In order to test the API manager product during each stage of pre- marketing process consistent background environments needed to be maintained. Therefore, developers try to re- create these APIs in a new environment. When there's no any clear defined mechanism to export and import the APIs created in the past environments, Developers has to set up and publish these APIs in the new environment manually. Manual creation of APIs causes several identified issues as below [1].

Additional developer effort requires in recreating the same API in different environments. Creating an API with minimal features in single given environment cost average of 2 minutes. Number of such APIs are created in the process of developing an API management products as WSO2 API manager, Apigee etc. re- creating these APIs in several other environments is a wastage of developer productive time.

Extra time spends on recreating APIs will drag the test schedules and thereby extend the final release days of the product. Further, it is a waste of productive time that can be used in any feature development tasks.

Loss of actual functionality and features. Even though the developers somehow manage to create the APIs in the new environments, some of the functionalities and the features need to be tested can be missed due to the lack of knowledge in building a full functional API as known by the individuals of the developing environment.

As a solution to address above mentioned issues APIs created in the developer environment needed to be a bundle and transfer to each environment. Only 39% of the total API managing products in the market have defined a mechanism to copy and migrate APIs across different environments. These API managing products have defined different approaches to perform API migrations. Still, some drawbacks have not been addressed as a whole in any of those migration mechanisms. Following are few such dominant drawbacks that reveal after the market research and the literature survey on current API import/ export processors [10].

Platform dependency. Most of the API migration mechanisms, migration tools are heavily relying on the built-in platform technology and cannot be accessed via other API managing products. This is mainly due to the tight coupling between tool's functionality and the programming language used in building the tool [2].

Delegation of major key security functions to third parties. Most of the currents migration tools rely on third party integrations to perform security functionalities directly on behalf on them. The trust relationship between the third-party application and the operating application is the foundation of this bond. APIs are proprietary properties. Therefore, security becomes a key factor that needs to address in advance. Sometimes this trust relationship can be broken and application can be open to intruders, therefore in order to withstand this circumvent alternative security mechanisms or stronger authentication techniques should be integrated into the tool [3].

Individual components should be deployed separately. Single API consist of a number of components as API definitions, swagger definitions, API thumbnails, documentation, WSDL and any migration policy sequences. In most of the migration tools each of these components needed to be deployed separately during the process of importing API to a new environment. This has a considerable effect on the performance of the migration process. In the process of bulk import and export this manual work increase in multiple times as per to a single API import and export. Ultimately, it's a waste of productive developer effort and time.

File exchange through third party applications. In the current process of exporting followed by few available API managing products, created APIs can be compressed in to .zip or into any other portable version and these archives been sent through a third-party software application like email, skype, google plus or using any other file sharing service. However, this procedure has open passage to intruder attack risks. It is better to handle the API migration through docker files which will be a perfect solution for unauthorized access via external parties.

Useful bandwidth wastage. When comes to the API management based on REST API implementations, Retrieving API's resources from data sources perform number of REST API invocations and same when storing components in-to the data sources of the destination environment. A number of REST API calls moving back and forth reducing the utilizable bandwidth allocated for the entire process.

Considering above facts, its showcase that there's unfilled requirement of a more explicit tooling support for the domain of API migration. The purpose of this

paper is to forward a preferable solution to the API migration process addressing above mentioned limitations.

III. METHODOLOGY

The objective of the project is to introduce an efficient and platform independent solution for migration of APIs across different environments. This tool is a generalized CLI tool, build for API migration which will facilitate the smooth and powerful transferring of APIs across different environments in the domain of API management. The CLI tool will minimize the effort and time in re-creating APIs in when product moves between environments as Dev to QA or QA to production.

Tool is a generalized migrating tool that can be used in different API management application by different vendors. Another main benefit of the proposed tool is, the tool will be a platform independent tool which can be used in different operating systems.

Tool comprised of several major key functionalities including built-in authentication mechanism, export of single and multiple APIs, import of APIs, API deployment in the google cloud using docker and kubernetes.

Security. The CLI tool equipped with a built-in authentication mechanism to ensure a strong secure authentication process. Therefore, it has minimized the dependencies with external third-party authentication mechanisms.

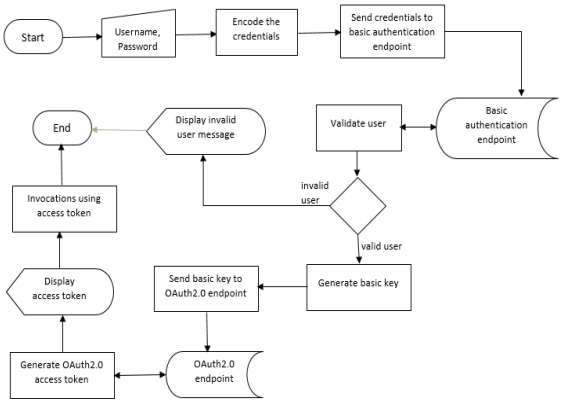

Figure 1: The Authentication mechanism of the CLI tool

Encapsulated authentication operations are undertaken to prevent any unnecessary external intruder actions. Figure 1 provides the detail flow of the authentication mechanism followed by the CLI tool. User can log in to the tool using a valid username and a password. This user credentials decide about the environments that the user can access into. Valid user name and a password then concatenated and encrypted and sent to the basic token endpoint. Basic token endpoint will issue a basic token with expiration period. This encrypted basic token can then be sent in to an OAuth 2.0 endpoint which will return a valid OAuth 2.0 token. OAuth token is use by the CLI tool to call the REST API endpoints to retrieve the resource components of the APIs. These components can then can be bundled and sent to the next environment.

Export. Retrieving all the components related to a API as API definition, swagger definition, documentation and bundle them up to a transportable file refers to as API export. The figure 2 illustrates the process followed by the CLI tool perform above functionality.

User can execute the tool with the credentials of the API details. API details can either be the UUID of the API or combination of API name, version and the owner of the API. Once the request been sent, CLI tool search for the API in API store. If it is a valid API CLI tool will retrieve all the components related to the API separately from the persistent data source and write in to a folder. Component belongs to API includes API definition, swagger definition, thumbnail images, mediation policies, documentations and any WSDLs if available. Finally created API been compressed and convert into a transportable file, which can be used in the API import process in the receiving end.

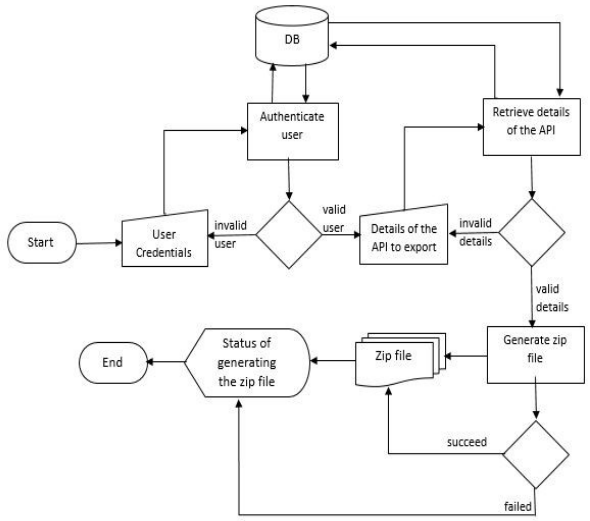

Figure 2: The work flow on exporting an API.

Import. In API import a publisher will receive a compressed version of an API or a collection of APIs. Publisher can create those into the new environment by executing the CLI tool. Figure 3 illustrates how the API import preform inside the CLI tool.

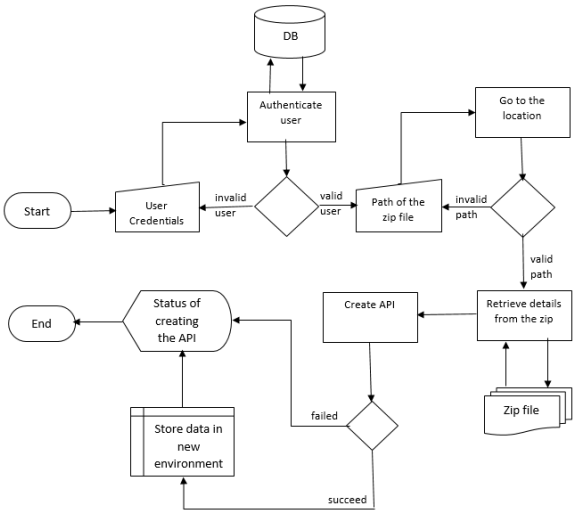

Figure 3: The work flow of the API import.

User can run the CLI tool with the valid credentials and path to the imported API file. Tool will validate the user and path to the imported file. If valid, CLI tool will extract all the content inside the imported file to a temporary location. There after tool execute number of REST API invocations to re-store the component in the persistent data source of the new environment. After storing data tool can publish the API/APIs in the new environment.

Deployment of API files using docker and kubernetes. The main communication mechanism used in REST APIs are HTTP calls, there can be many numbers of HTTP calls as per the user's requests. It will be affect the performance of the product while wasting a certain amount of network bandwidth. As a solution, introducing a way to minimize the number of HTTP calls to achieve the use of a minimal number of resources per requests to improve the performance and accuracy of the tool. Moreover, including Docker in the product to make more efficient service since Docker has many potential advantages in the fields of usability, performance and security against traditional virtualization [4],[8]. Base Docker image from the Docker hub will be taken through the CLI tool and then push the API to the retrieved Docker image. After that, it can be pushed directly to the Docker hub again. Another production environment can access the pushed Docker image with the API in their environment to execute the tasks. Therefore, Docker makes easier to deploy CLI tool in several isolated environments [6], [7]. Always there may minor variation between development environments; unless having own repository environment. By using Docker, fulfil that gap by keeping consistent environment because Docker containers are configured to keep dependencies internally.

Furthermore, involving kubernetes in the tool will also benefit the user, since it can handle complex scenarios on the deployment and to give users the ability to access their API's with more efficient and scalable approach while providing zero downtime deployments, continuous deployment and high stability of deployed services [5],[9]. Therefore, the purpose of Kubernetes is to make it easier to organize and schedule the application across a fleet of machines. At a high level, it is an operating system for the user's cluster. It handles what specific machine in the datacenter each application runs on.

The configurations needed to be done at the user end will be minimized by the above-mentioned techniques by enhancing the automation functions which will be included in the proposed tool. The final implemented CLI tool will mainly achieve API migration in

different environments, platform independent, high efficiency, provide powerful built-in authentication mechanism and minimalist usage of the available bandwidth while minimizing the identified drawbacks and improve the performance of current migration tools in API management products.

IV. Conclusion

API migration is becoming an essential functionality supported by API managing products which will allow developers to migrate the created APIs in one development environment to another as well for the API publishers to exchange their created APIs with other API publishers. Current mechanisms provided by several API management products have identified problems which are not addressed yet. Therefore, a requirement for a powerful, platform independent and efficient CLI tooling supporting for this domain is still at a growing stage. We presented new CLI tool which could overcome those identified issues in current existing tools and have a capability to address the performance issues currently undergoing.

Acknowledgment

THIS RESEARCH WORK SUPPORTED BY SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY.

References

[1] Alexie Balaganski, “API Security Management”, KuppingerCole Report No : 70958, LEADERSHIP COMPASS pp 20-27, July 2015.

[2] Victor Winter, Jonathan Guerrero, Carl Reinke, James Perry,” Java Core API Migration: Challenges and Techniques”, in PPPJ 2013 submission, 14 May 2013.

[3] Trevor Perrin, Logan Bruns, Jahan Moreh, Terry Olki, “Delegated Cryptography, Online Trusted Third Parties, and PKI”, in 1st Annual PKI Research Workshop - April 2002.Available: http://trevp.net/delegatedCrypto/delegatedCrypto.html. [25 May 2017]

[4] Tuomas Vase, “ADVANTAGES OF DOCKER”, University of Jyväskylä, 2015, 24 p. [Online] Available: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/48029/URN%3ANBN%3Afi%3Ajyu-201512093942.pdf?sequence=1, [May.25 2017]

[5] Jean-Mathieu Saponaro. (2017, May 19). “Kubernetes”. [Online]. Available: http://blog.kubernetes.io/, [May.25.2017]

[6] Docker Doc,“Overview of Docker Hub”.Internet:https://docs.docker.com/docker-hub/ [May.25,2017]

[7] Muhammad Arul.” How to create Docker Images with a Dockerfile”.Internet:.https://www.howtoforge.com/tutorial/how-to-create-docker-images-with-dockerfile/ [May.25,2017]

[8] IOD Community.” 5 Key Benefits of Docker: CI, Version Control, Portability, Isolation and Security”. Internet: http://www.iamondemand.com/blog/5-key-benefits-of-docker-ci-version-control-portability-isolation-and-security/ .APR.24, 2015[May.25,2017]

[9] Wikipedia. “Kubernetes”. Internet: https://en.wikipedia.org/wiki/Kubernetes,Mar.11, 2017. [May.25,2017]

[10] Alexie Balaganski. (2015, July). “Leadership compass”. API Security Management. [Online]. pp 20-27. Available: [May.27 2017]