API Fabric Project

Update September 2019: A PoC for demonstrating the capabilities of API Fabric is being developed. This will be demonstrated by mid-September. Please refer to the child page for details.

Update October 2019: API Fabric proposal is put on hold and to be revisited after the Frankfurt release cycle.

Supported Operators: Vodafone, Swisscom (In discussion with other Operators)

Project Name

- Proposed name for the project: API Fabric
- Proposed name for the repository: apifabric

Project Goal

Component for managing high-level APIs exposing the business logic of ONAP. Handles API management including the API Lifecycle, API Monitoring, API Security and Policy Control, API Transaction Logging, API Abstraction, and Transformation. Currently, there is a module in the MSB project with name Internal/External API Gateway. There is also an External API project. with the responsibility to integrate ONAP with OSS/BSS Solutions through TMF APIs. The proposed functionality should not be confused with what MSB or Ext-API is providing as of today and suggested to check the comparison below with Ext-API and MSB.

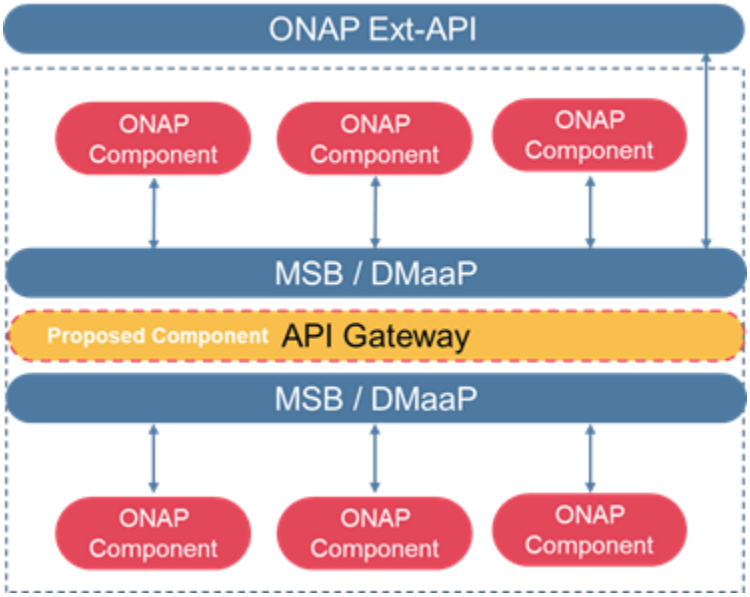
Background

ONAP today has many components/projects and many APIs exposing functional capability of components. There are dedicated components which manage API transformation between internal and external systems - such as Ext-API, Multi-Cloud, SDNC/App-C etc. There are also projects which compose functionality and expose low-level APIs (which are focused on exposing component specific management/functional interfaces). Most of the times, such low-level APIs are very complex to manage and consumer (such as other components in ONAP, external applications, use case specific applications etc.) needs to be aware of very minute details (version, certificate, authentication, the right entity to be operated on, etc) of the provider to use the APIs in the right sequence and with the right information. This often leads to the tight dependency between components and additional overhead for project teams. There is no logical layer which consolidates these APIs and exposes a façade that is consumer friendly and hides many complexities associated with component level functional APIs. The need for such a facade layer has been discussed many times in the past in relation to the Modularity and also to make ONAP production ready/standard compliant etc. The API GW Project is proposed to address the following problem statements

- **Different API Management approaches**: In ONAP each component exposes own APIs corresponding to the functionality supported. In addition to this, each project follows different approaches for API security, API documentation, and moreover API style itself (entity based, action based etc). This creates a lot of burden for the API consumer, especially use case developer who is focused on the high-level functionality to be leveraged than component level API intricacies (version, security, style, etc.). There should be consistency across components in defining APIs or there should be a logical layer (Facade) which hides these inconsistencies/complexities so that the API consumer need not build different set of adaptors or require different expertise to consume the component level APIs.
- **Standard Alignment is a priority in ONAP**: In recent releases, standard interface and model alignment became a hot topic of discussion. Currently, standard API alignment is addressed by individual projects independently depending on specific use cases. There is also the redundant implementation of adaptors – for example, SOL005 API adaptor proposed for SO/Ext-API , SOL003 API Adaptor in SO for S-VNFM connectivity, SOL003 adaptor development in VFC, SOL005 adaptor development in VFC NFVO, SOL003 adaptor in SDNC and SOL002 adapter in SO, etc. There should be a consistent adaptation layer that can be consumed by internal and external API consumers so that maintenance and compliance of the APIs can be managed well.
- **Production deployments require interoperability with legacy and 3rd Party components**: In typical production deployment of ONAP, it will require integration with legacy and 3rd Party applications in the operator premise. Currently, the expectation is that System Integrator responsible for the deployment build appropriate integration layer and enable interoperability with such external applications. But this is not a healthy and consistent solution as ONAP component level interfaces evolve over releases and for applications, this becomes unmanageable as these ONAP interface APIs have some times release specific dependency and non-backward compatible. There should be a logically centralized function in ONAP which can abstract component level API evolution and expose a higher level API as well hides the dynamics at the individual component APIs. Additionally, integration with 3rd party applications in productions environment might require different types of policy enforcement, different types of adaptations. In production deployments, ONAP components may not be compatible with operators IT systems - for example, the operator might already have an Identity Provider and Auth Provider which are not compatible with AAF. Such scenarios might require additional capability in ONAP where these incompatibilities can be accommodated rather than instrumenting each ONAP project. Another requirement is related to Modularity - Not all components in ONAP might be required in the Operator production environment, depending on the evolutionary stage of transformation. They might have already procured redundant functions as in ONAP that fulfill the use case requirements. This leads to a requirement where different ONAP components (not in full but selectively) need to be composed at different levels of abstractions and integrated with existing solutions. One such scenario is Legacy NFVO integration with ONAP as discussed in the Orchestration Scenarios task force- Similar requirement would require special integration which if handled at individual ONAP project may soon become unmanageable.
- **Evolution of platform functional capability vs use case based functional enhancements** : This is a typical problem in most of the open source projects - the area the project should focus – i.e. to focus on developing functional capability and make it more generic for any use case to adapt or to enhance functional capability based on specific use cases. In ONAP the functional capability of projects is primarily driven by use cases. The requirements for each use case might be different or might require a different level of capability enhancements at the project level. For project teams, this creates lot of pressure to balance requirements across use cases and also to focus on progressing the functional capability for long term requirements. There should be a logical boundary where both such requirements - from use case or from project functional enhancement – should balance. The boundary should be an API layer which cushions the impact and hides the project level evolutions at the same time supports current and future use cases.

Project Description

The proposed project – API GW, acts as an API broker between external/internal components in ONAP and provides logically centralized API management and control capability.



API GW closely works with three components in ONAP – Ext-API , MSB and DMaaP. API GW can provide a proxy interface to Ext-API project and enable secure connection, Federation, Policy enforcement, load balancing, and overall API Management capability. API GW can enhance the capability of MSB with built-in plugins that enhance MSB to have capabilities such as API LCM, Marketplace, API Catalog management, Analytics, RBAC, Security Management, API Aggregation and Composition, Script insertion, Expression language support, etc. API-GW can act like a DMaaP client and expose notifications to external applications through a subscribed hub resource.

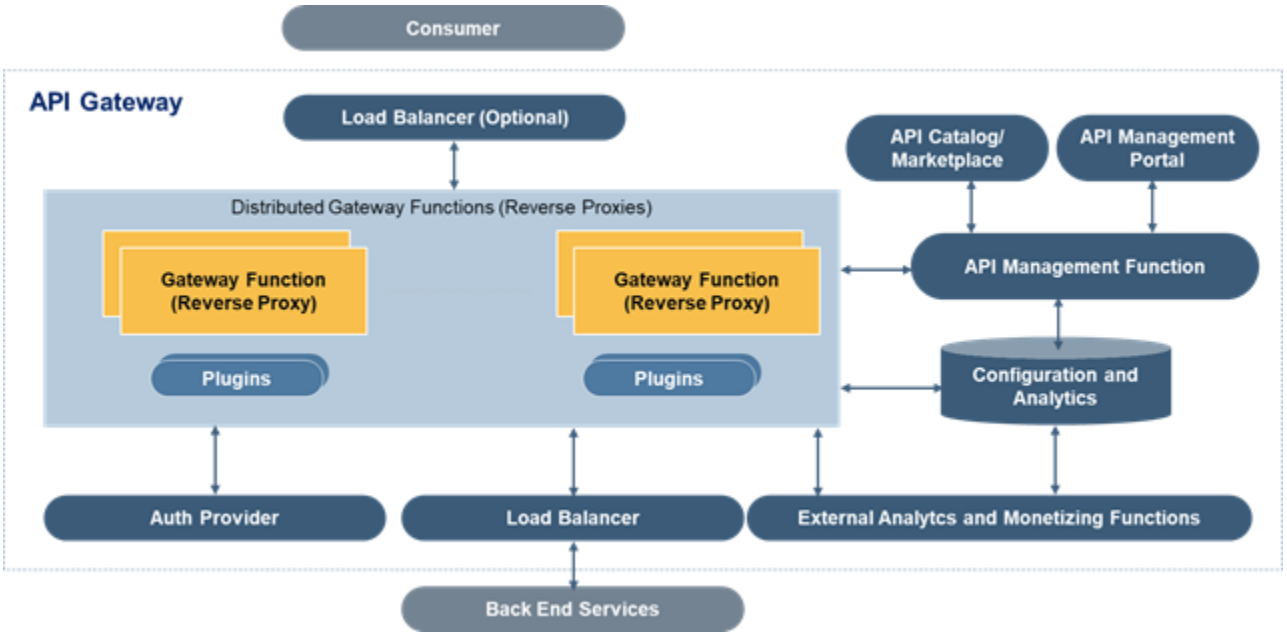
This proposal does not intend to replace any of the existing components/functionality in ONAP. Some of the redundant functionality that exists between API GW and ONAP Components such as Ext-API and MSB can be managed through collaboration and mutual agreement. API GW can also potentially be a subproject of either Ext-API or MSB to augment the missing capabilities in respective projects. Another potential option is to merge functionality in Ext-API, MSB, and API-GW to create a single API Management/Adaptation component.

Please note the comparison of API GW and existing ONAP projects like Ext-API and MSB below. The comparison also covers what additional capability API GW bring in to augment the existing functionality. As for this proposal, the directions provided by Architecture committee and TSC will be followed once the need for such functionality is agreed with all stakeholders.

How API GW Solution solves the problems stated?

- **Different API Management Approaches:** Logically Centralized API GW component enables consistent approach for managing APIs without the need to develop redundant API Adaptor, hiding a variety of approaches followed by different ONAP components
- **Standard Alignment:** API GW support rich transformation capability with inbuilt plugins and support for expression language that can be configured on demand rather than developing from scratch (and subsequently waiting for a long release cycle.)
- **Production deployments might require interoperability with legacy and 3rd Party components:** This is an inherent capability of API GW. API GW allows flexible integration with external and internal systems with toolsets for HTTP Method conversion, HTTP Payload Conversion, REST/SOAP transformation, JSON/XML transformation, JSON to JSON transformation, Different types of security mechanisms, etc. API GW can expose Façade APIs which compose the internal functionality of ONAP at desired levels of complexity**.**
- **Evolution of Platform functional capability vs. Use Case capability:** Helps in creating abstraction layer or Façade as needed for different use cases, allowing the projects to evolve independently

Architecture



API GW consists of four main building blocks – Gateway, API GW Management Component, API GW Configuration store, API GW UI. Gateway function is a stateless reverse proxy which can be instantiated on demand, scaled and distributed. It hosts a set of plugins that enhance the API control logic. The plugins are developed using the gateway SDK and attached as a library during the initialization. Gateway and Plugins refers the API configuration (i.e information about the on boarded API) from Configuration and Analytics data store. The Configuration & Analytics DS persists the API configuration and API transactional metrics. API GW can work with an inbuilt Auth provider capable of centrally managing the Authentication, Authorization of the exposed APIs. The API GW UI hosts the Management, Design and Monitoring UIs. The GW function maintains statistics and log information of APIs and stores the information in the Configuration and Analytics store. It is also possible to integrate the Configuration and Analytics Store with external monitoring solutions like Prometheus or alert engines to notify external consumers about API statistics.

Scope

Following are the proposed capabilities

1. API LCM: Manage the lifecycle of API – Onboarding of APIs as swagger or equivalent templates, Design of APIs with context path and backend endpoints, Association of API with required plugins to control runtime behavior, Activation or Deactivation of API, Management of Security aspects of the API
2. API Market Place, API Catalog Management: Provide a consumer-friendly and developer friendly API Management interface
3. Plan, Subscription Management: Ability to design API plans with different levels of control and provide management interface to subscribe to a particular API and approve the specific subscription.
4. Content/Payload based API routing: Currently MSB in ONAP Supports API routing based on Service endpoints, but not strictly based on the API payload or API headers. This is an augmented capability on top of MSB to be consumed by use case teams and projects so that any custom routing can be incorporated at the API layer rather than at the individual project level.
5. API Federation: Currently there is no consistent mechanism for federation between two ONAP instances at least at the API layer. This capability allows projects to leverage API GW as a gateway that manages communication between multiple instances of ONAP. Note that this capability only covers the
6. Consistent Security Management: Manage API security– primarily transactional security – Authentication, Authorization, Encryption. Authentication and Authorization is planned to be based on a pluggable model, i.e capability to integrate with the Auth Provider IDP within the operator premise or reuse capabilities provided by AAF. Encryptions is enabled through HTTPS based secure channel – for this API GW maintains a keystore and truststore (PKCS12 based stores) with certificates signed by an authorized signing authority. API GW supports Authorization/Authentication based on OAuth 2.0, Open ID Connect, SCIM 2.0, etc.
7. Circuit Breaking, Timeout, Retries, and Rate Control: Capability to control the API consumption by tuning the APIs usage properties. These capabilities may be controlled on demand using the management APIs exposed by API GW.
8. Flexible Request and Response Transformation: Capability to transform the API payload (request or response) based on predefined transformation logic – The transformation logic can be implemented as plugin and can be configured at runtime using exposed properties. The transformation plugin may also support expression languages (such as JOLT) or script insertion to transform the request or response. Other transformation capabilities include header, URL transformation, XML/JSON payload transformation, Request Method transformation, Security mechanism transformation etc.
9. API Sharding (Targeted API Deployment) : Targeted deployment of APIs at distributed API GW Instances based on specific criteria – i.e geographic proximity, load etc.
10. Service Discovery: Discover the API endpoints (backend Service APIs) based on registry look up and load balance the request across discovered services.
11. Façade: Aggregate/Compose complex low-level APIs and expose simplified façade APIs associated with service endpoints
12. API Policy Enforcement: Define API control policies – security or run time behavior
13. Common look and feel and documentation: Ensure common look and feel, documentation for exposed ONAP APIs
14. API Metrics Collection, Analytics, Metering, Audit, Logging: Capability track all the API transactions and identify the usage pattern, traffic
15. Alerts: Enable API consumption specific alerts so that corrective actions can be carried out on-demand.
16. White Listing, Black Listing of APIs based on the Subscription profile, Policy

17. API Designer: a Designer tool to import swagger APIs, attach appropriate policies associated with API, commission and decommission APIs, manage subscriptions and plans

Short Term Capabilities (Proof of concept and first release after completing PoC):

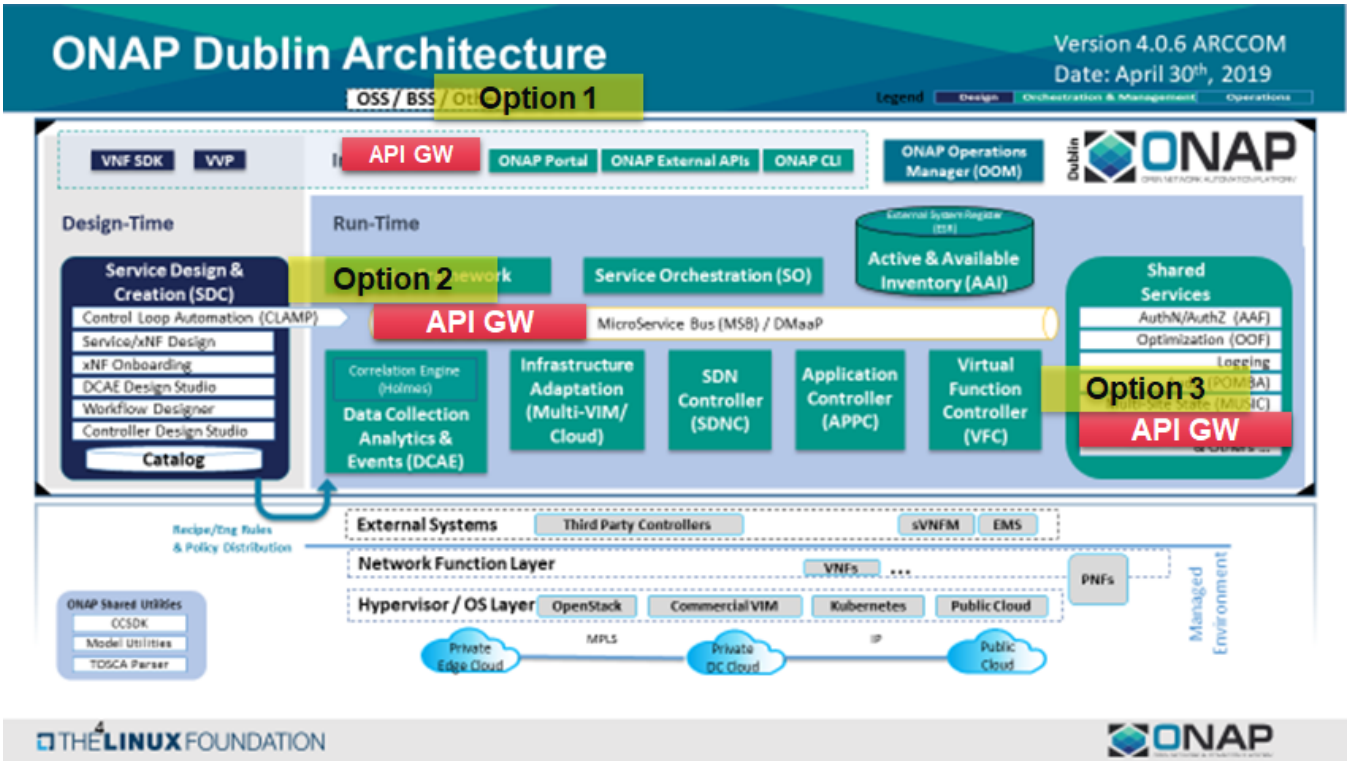
- Capability marked in Blue (To be prioritized)

Long Term Capabilities:

- Rest of the capabilities not covered in Short Term

Architecture Alignment

Potential placement of API GW in ONAP Architecture



Option 1: Co-exist with Ext-API , but may support external and internal APIs on need basis

Option 2: Co-exist with MSB, but handles gateway functionality independently. MSB handles the Registry and Service Discovery.

Option 3: API GW exists as an independent functional component

API GW and External API:

- Exiting Capabilities in Ext-API :
  - Mediation/Adaptation between TMF APIs and ONAP internal APIs
  - Leverages JOLT JSON Transformation Templates for Payload transformation
  - Order State Monitoring – Hub Resources Management for callbacks
  - Repository for Service Specification Catalog , Service Order Mapping details
  - Leverages SDC JTOSCA Parser for TOSCA Parsing
  - Static transformation logic and routing implemented in code
- Capabilities Augmented by API GW Solution
  - Management toolsets for configuring API context and endpoint
  - API Analytics
  - Full API Lifecycle Management – Onboard, Policy Control, retire, WL,BL
  - API Subscription/Plan management
  - API Policy management
  - Enhanced API Security Management – OAuth2, JWT, Open ID, SCIM 2.0 – All inbuilt and centrally managed
  - Script insertion in API execution flow
  - Configurable APIs, Transformation logic than static Code
  - Pre-built API Processing plugins
  - API Aggregation and Composition

    - Swagger Import and Plugin chaining
    - Management and Monitoring UI

API GW and MSB:

- Existing Capabilities in MSB
    - API Endpoint Registration and Discovery
    - Static API Endpoint Routing based on port and Service URL (No payload-based routing)
    - API Load balancing
    - Service Mesh Integration Prototype
    - Integration with AAF for security policy enforcement (?)
    - Integration with OOM for dynamic Service Registration
    - Management APIs for registration of Services
- Capabilities Augmented by API GW Solution
    - Full API Lifecycle Management
    - Manual and Bulk API Import – Swagger or Management API
    - API Subscription/Plan management, API Discovery
    - API Catalog and Marketplace
    - Integration with multiple external IDP, Monitoring solution
    - Rate Limit, Quota Mgmt , Circuit Break
    - Tenant, Role Management
    - Whitelisting , Black Listing
    - Enhanced API Security Management – OAuth2, JWT, Open ID etc. – All inbuilt and centrally managed
    - Script insertion in the API execution flow
    - Configurable APIs, Transformation logic
    - API Aggregation and Composition
    - Management and Monitoring UI

Benefits for ONAP

- Support a single source of truth for High level APIs, rather than each project maintaining own versions
- Augments MSB and Ext-API capabilities
- Facade layer: Enables development of a Facade layer which abstracts the complexities of internal API
- Request/Response Transformation: Enables ONAP components to align with SDO APIs more easily without changing the existing capabilities
- Low impact on existing projects: Enable Operators to plugin standard and legacy integration API adaptors without impacting the ONAP components
- Allows Projects/Components to focus on core functionality rather than worrying about API Transformation
- Enables Tenancy/RBAC Management: Centralized API management can help in the implementation of tenancy management through policies.

- **How does this project fit into the rest of the ONAP Architecture?**
  API Gateway provides an API and UI interface. API can be used by components in ONAP or by administrative users to manage the lifecycle of high-level APIs. UI is primarily intended for an administrator to provision and manage APIs. The UI is also suitable for the end user as the API GW provides a consumer view – API Marketplace which can be used to subscribe to specific APIs.
- **What other ONAP projects does this project depend on?**
  OOM for deployment, MSB or Specific ONAP components whose APIs need to be abstracted behind a façade API (standard API), AAF for Security – Authorization, Authentication and Certificate Management
- **In Relation to Other ONAP Components**

- External API
- MSB

- **How does this align with external standards/specifications?**

- APIs/Interfaces - REST, JSON, XML
- Information/data models - Swagger JSON

- **Are there dependencies with other open source projects?**

- MongoDB
- Gravitee/Kong/Gloo
- Elastic Search

Other Information

- link to seed code (if applicable)

NA

- Vendor Neutral

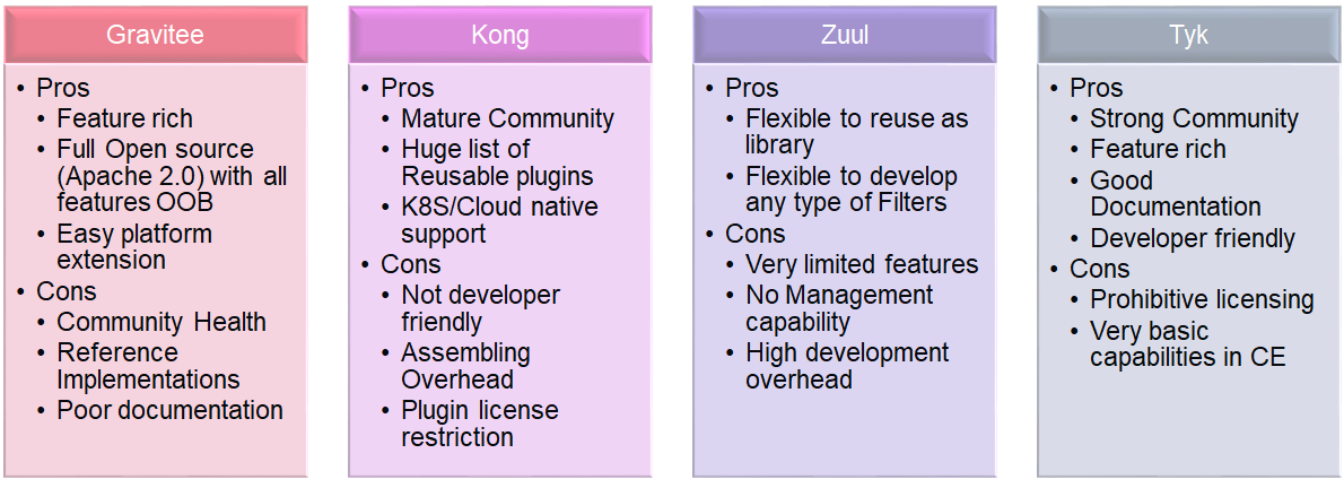
NA

- Meets Board policy (including IPR)

NA

- *Note on Open Source Solution:*

Plan to develop Project PoC using the Gravitee API Gateway (https://gravitee.io/) Solution which is cloud-native, developer-friendly and features rich. However, we are also open for adapting other API GW solutions such as Kong or Gloo. Kong (link) is based on Nginx & OpenResty and have similar capabilities like Gravitee but requires the extension/plugins to be written in unpopular Lua script, additionally, it has limited reusable libraries to be used for developing plugins. Gloo (https://github.com/solo-io/gloo) is a project endorsed by CNCF and has a dependency on Envoy proxy, so can be considered when ONAP migrates to a service mesh based microservice communication model. Following is a high-level comparison (Note: our current subjective view based on the scope defined above) of different API GW solutions that we have evaluated. Inputs/suggestions are welcome from community members on other alternate open source solutions that we can leverage with friendly licensing terms.



Note on S3P :

- Scalability: Depends on the particular open source solution being selected - Majority of the open source API GW solutions support distributed scaling.
- Security: To be discussed with the Security Subcommittee - API GW inherently supports rich security management features - OAUTH2.0, Open ID Connect, Certificate management etc.
- Stability: All the proposed features are from mature API GW open source solutions which are in the production environment. Guidance from Architecture team on specific areas awaited.
- Performance: For API GW two aspects of performance is critical - Latency and Num of Requests per Second. Latency depends on the complexity of API and associated nested operations that need to be carried out for a particular API. The number of requests handled by API GW depends on the scalability features provided by the API GW. Performance benchmark can be carried out based on the requirements set by the S3P team or specific use cases.

Key Project Facts

Facts	Info
PTL (first and last name)	TBD
Jira Project Name	APIGateway
Jira Key	APIGATEWAY
Project ID	apigw
Link to Wiki Space	TBD

Release Components Name

Note: refer to existing project for details on how to fill out this table

Components Name	Components Repository name	Maven Group ID	Components Description
apigw	apigw	org.onap.apigw	Component for API Design, LCM , Monitoring, Control

Resources committed to the Release

Note 1: No more than 5 committers per project. Balance the committers list and avoid members representing only one company. Ensure there is at least 3 companies supporting your proposal.

Note 2: It is critical to complete all the information requested, that will help to fast forward the onboarding process.

Role	First Name Last Name	Linux Foundation ID	Email Address	Location
PTL	TBD			
Committers	Manoj Nair (NetCracker)	Mknair75	manoj.k.nair@netcracker.com	Bangalore, India, GMT+5:30
	Ramesh Iyer(NetCracker)			Bangalore, India, GMT+5:30
	TBD			
	TBD			
	TBD			
Contributors				
	Abinash Vishwakarma (NetCracker)			
	Andrei Chekalin (NetCracker)			