ONAP
OPEN NETWORK AUTOMATION PLATFORM

ONAP API Gateway Proposal

Proposed by：NetCracker Technology
Supported by : Vodafone, Swisscom (Discussion with other operators in progress)

07th May 2019

Summary

Management of Internal/External integrations between Components is a traditional pain point of ONAP Platform (Integrations with 3rd party systems, Internal integration over Standard APIs, OSS/BSS Integration etc.)

The study defines a problem statement for current situation of API Management in ONAP, and corresponding proposal for starting a new ONAP Project (or work with other Projects to fix the gaps)

Agenda

- Problem Statement
- Proposal
- Execution plan

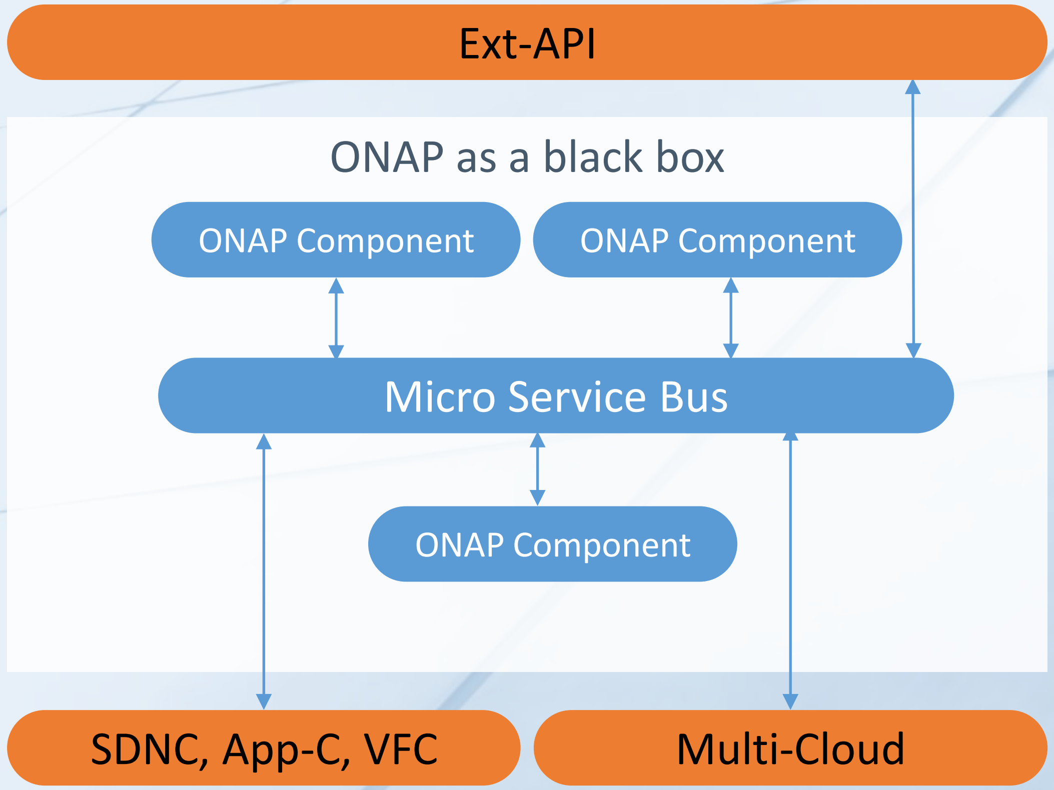
Section 1

Problem Statement

API Management – What is Available Today

A Set of ONAP Components focused on API Mediation/Adaptation & Routing

- **External API** treats ONAP as a black box and exposes NBI, transforms standard (TMF) NBI to ONAP internal API
- **Multi-Cloud** used for mediating/abstracting interactions across multiple cloud environments
- **Controllers** (VF-C, App-C , SDNC): Mediates the LCM of Virtualized Resources and supports Configuration Management in a vendor neutral way
- **Micro Service Bus** routes, Load balances internal API calls to registered end points



Problem Statement

Zoo of API Management Approaches:

- APIs are managed at individual project level : Each component exposes very low level capability , not all will be necessary always to represent the business logic
- API consumer is depended on the component level API intricacies, Entity model rather than what is necessary and sufficient
- No consistent approach across projects (security, documentation, Version compatibility, style etc) in managing APIs

Standard Alignment is a priority in ONAP

- Enhancing multiple components for standard API alignment is time consuming
- Redundant API adaptation logic across different components that cannot be reused – e.g. SOL003 adaptor in SO , VFC and SDNC
- Overhead on project teams to manage standard adaptation than core functionality

Production deployments might require interoperability with legacy and 3rd Party components

- Need for an API abstraction /façade layer rather than point to point integration with each component
- Capability to compose APIs exposed by different components at different levels of abstraction and integration with 3rd party , External Components

Evolution of Platform functional capability vs. Use Case capability:

Platform need to evolve independently, not strictly based on use cases:

- Missing an appropriate facade layer to isolate these two needs
- Use cases typically expect standard/composite APIs for wider acceptance and adoption, project specific API alignment roadmap not completely in sync with use cases and delay the use case development.

What is Required?

Centralized API Management / Gateway Function

- A function/framework to build API Façade that gives flexibility/features for following
  - Model Driven : Import/Export high level APIs as Swagger file (Not code developed from scratch)
  - **API LCM, API Market Place , API Catalog, Plan, Subscription Management**
  - **Compose/Aggregate and expose simplified façade APIs for internal service end points**
  - Content/Payload based API routing
  - **API Federation across SP/Partner/Opco ONAP instances with desired policy enforcement**
  - **Flexible Security Management** (OAuth2.0, Open ID, SSL/TLS, Ext Auth provider integration)
  - Circuit Breaking, Timeout, Retries, Rate Control
  - **Flexible Request and Response Transformation**
  - API Sharding (Targeted API Deployment)
  - Service Capability Discovery (i.e. in addition to URL end point)
  - **Standard adaptors for transformation (between SDO API and internal API)**
  - **API Policy Enforcement**
  - Common look and feel and documentation
  - **Analytics, Metering, Closed and Open Loop Control of APIs**

Section 2

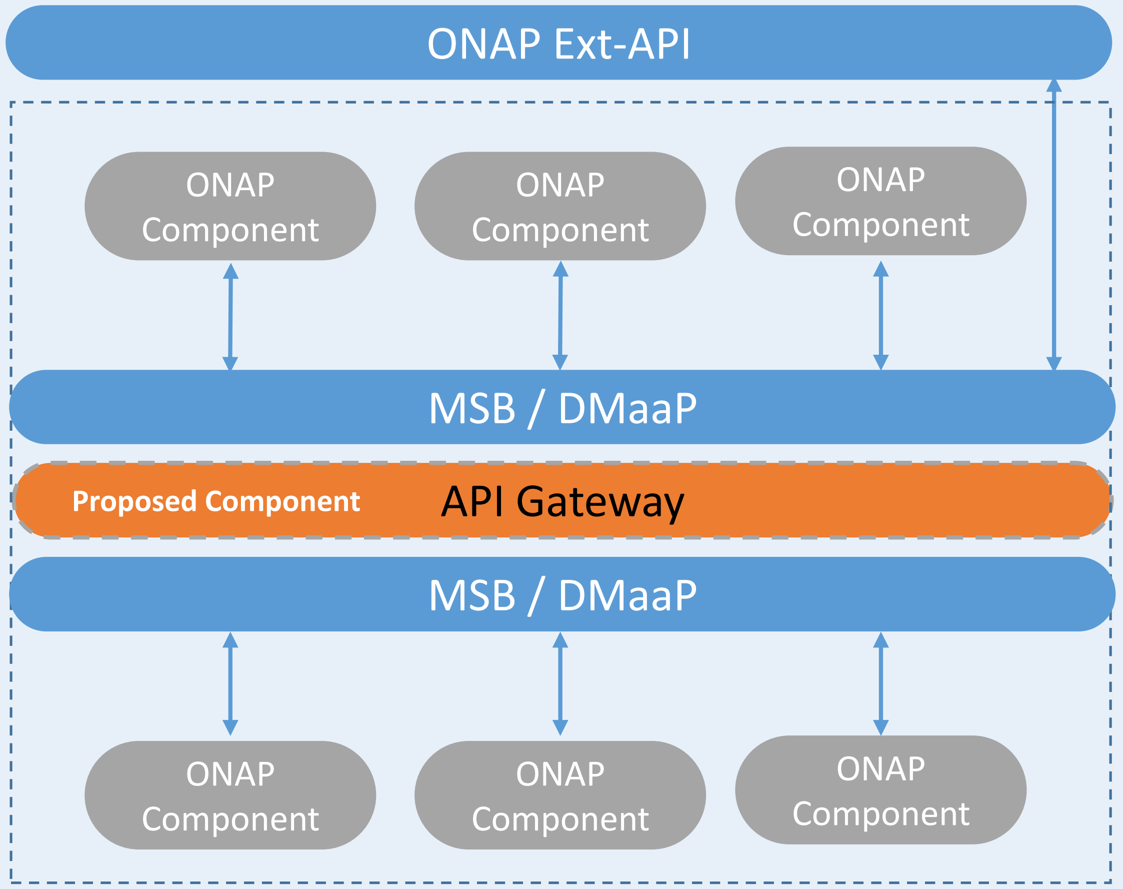
Proposal

Proposal: A dedicated API Gateway Function

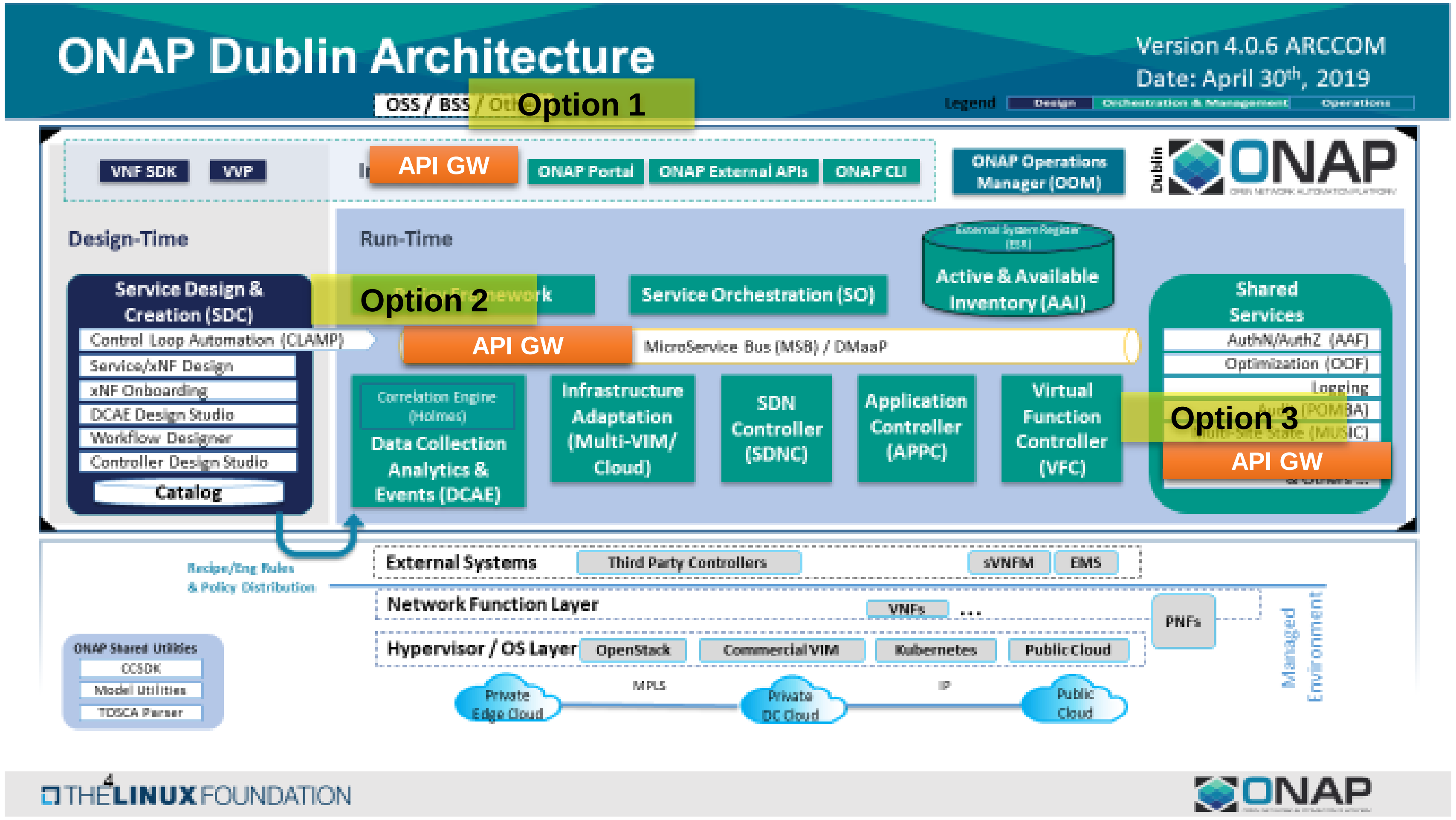
A new function that is dedicated for managing high level APIs across components.

FEATURES:

- Consolidates API Management in a single logical function
- Augments Integration Layer capabilities in ONAP
- Reuse API Routing Functions available in MSB
- Supports Plugin model to attach request and response transformation logic
- Offload common API tasks from other ONAP Components (ex. authentication and aggregation)
- Reuse open source solutions like Kong/Tyk/WSO2/Zuul/Gravitee/Gloo



API GW Placement 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 ONAP External API

FUNCTIONAL 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

FUNCTIONAL CAPABILITIES AUGMENTED BY API GW

- 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 Connect etc. – 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 (API Orchestration)
- Management and Monitoring UI

- External API is close to 30K lines of code and all API adaptors developed from scratch (required custom transformation and enrichment)
- Difficult to manage in the long run – need to leverage a specialized API GW function which can leverage built in plugins and transformation tools

API GW and ONAP MSB

FUNCTIONAL CAPABILITIES IN MSB	FUNCTIONAL CAPABILITIES AUGMENTED BY API GW

FUNCTIONAL CAPABILITIES IN MSB

- API End point 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 and Discovery
- Management APIs for registration of Services
- Basic MSB UI
- Web socket support

FUNCTIONAL CAPABILITIES AUGMENTED BY API GW

- **Full API Lifecycle Management**
- Manual and Bulk API Import – Swagger or Management API
- **API Subscription/Plan managemen**t
- API Catalog and Marketplace
- **Integration with multiple external IDP, Monitoring solution**
- **Rate Limit, Quota Mgmt , Circuit Break**
- **Tenant, Role Management**
- White listing , Black Listing
- **Enhanced API Security Management – OAuth2, JWT**, **Open ID etc. – All inbuilt and centrally managed**
- **Script insertion in API execution flow**
- Configurable APIs, Transformation logic using expression language
- **API Aggregation and Composition**
- Management and Monitoring UI
- GraphQL support

- MSB is built on NginX and OpenResty with additional plugins. Though MSB has pre-built API Gateway functionality – External and Internal API Gateways – These are limited in functionality and not used well in ONAP .
- Existing plugins focus on Routing and Service Discovery – Not providing full functionality offered by typical API GW
- MSB plugins built on Lua script and requires learning curve. Additional development overhead for new plugins and API LCM
- Suggestion is to leverage a full fledged API GW open source solution with OOB capabilities and build MSB capabilities in that.

API GW and DMaaP

FUNCTIONAL CAPABILITIES IN DMAAP

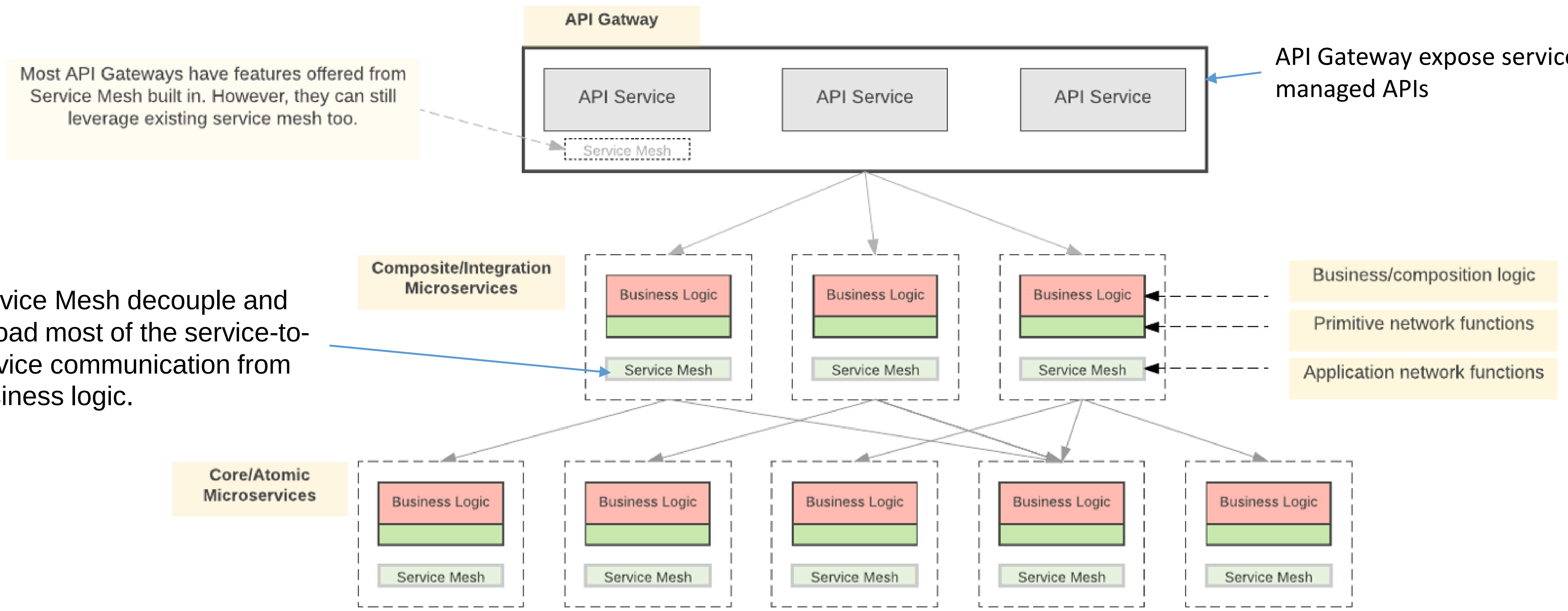
- Event Publish/Subscribe Mechanism
- Manage Topics – CRUD Operation
- Manage Subscriptions
- Provide a Façade API over Kafka Message Bus
- Client SDK for Working with DMaaP
- Distributed Deployment

FUNCTIONAL CAPABILITIES AUGMENTED BY API GW

- Asynchronous Event Notifications to API Consumers
- Offer Consumer Specific Adaptation for internal Events
- Offer a Web Socket or Server Sent Events Interface to Consumers for internal Events
- Pre and Post API Invocation notifications to ONAP internal components

- API GW does not have any conflicting capabilities with DMaaP. The two components are complimentary
- API GW can act like an external notification point by registering call back subscriptions to specific topics in DMaaP
- API GW will reuse DMaaP through a custom plugin (which use DMaaP Client SDK)

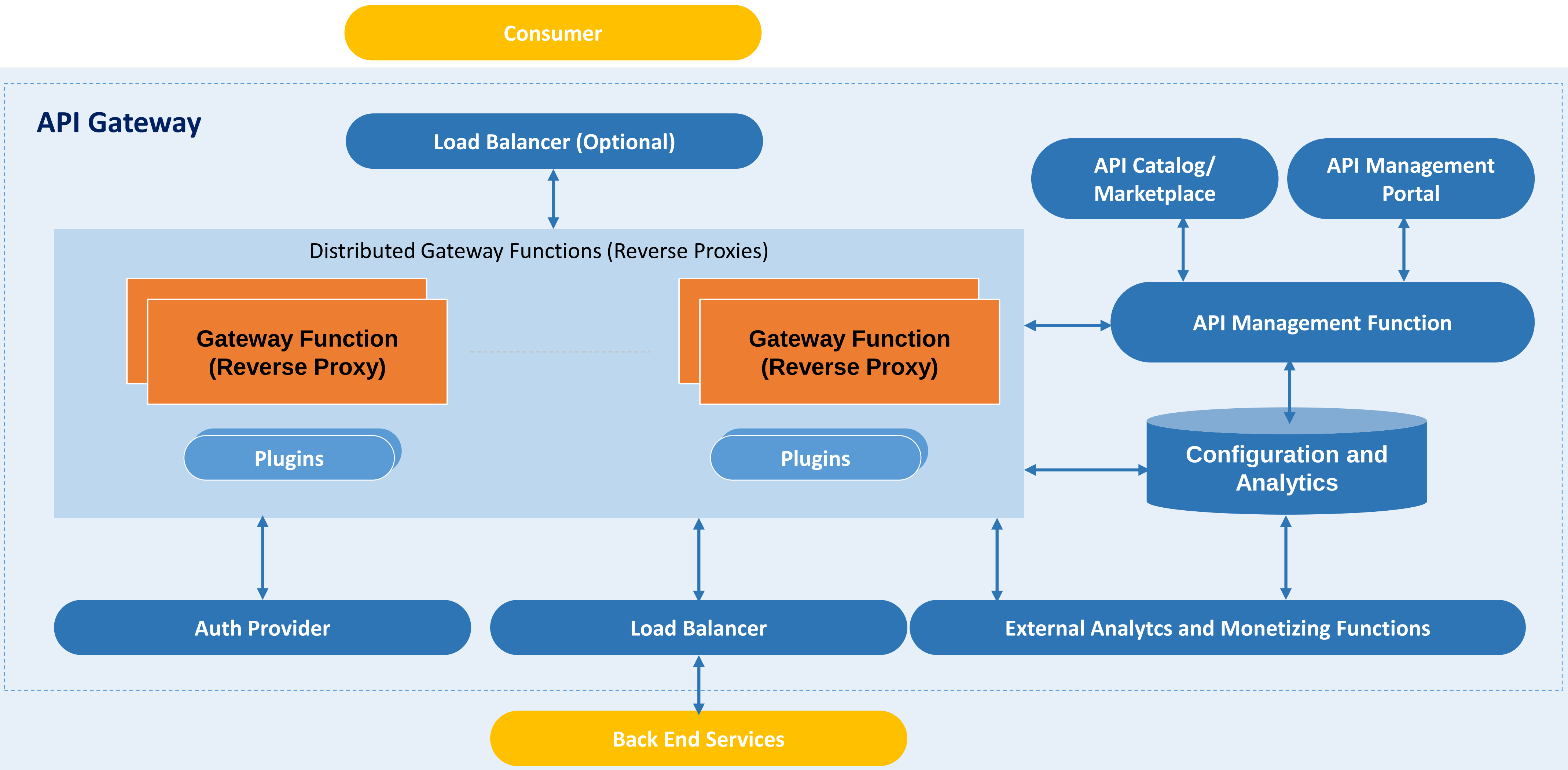
API Gateway and Service Mesh



API Gatway
Most API Gateways have features offered from Service Mesh built in. However, they can still leverage existing service mesh too.
API Service
API Service
API Service
Service Mesh
API Gateway expose services as managed APIs
Composite/Integration Microservices
Business Logic
Service Mesh
Business/composition logic
Primitive network functions
Application network functions
Service Mesh decouple and offload most of the service-to-service communication from business logic.
Core/Atomic Microservices

Service mesh is an inter-service communication infrastructure <u>which doesn't have any business notion</u>. So it will be ideal to be used at levels of Microservices.

Typical API GW Functional Architecture


Consumer
API Gateway
Load Balancer (Optional)
API Catalog/ Marketplace
API Management Portal
Distributed Gateway Functions (Reverse Proxies)
Gateway Function (Reverse Proxy)
Gateway Function (Reverse Proxy)
API Management Function
Plugins
Plugins
Configuration and Analytics
Auth Provider
Load Balancer
External Analytcs and Monetizing Functions
Back End Services

Benefits for ONAP

- ✓ Support single source of truth for Standard APIs, rather than each project maintaining own versions
- ✓ Augment MSB and Ext-API capabilities:
  with Request/Response Composition, Filtering, Policy Enforcement, Security, Orchestration
- ✓ 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 Management : Centralized API management can help in implementation of tenancy management through policies.

Section 3

Execution Plan

Proposed Plan

- **April-May** : Presentation to Operators in ONAP community and see if there is any need for such functionality – **Already presented to more than 6 operators in ONAP Community. Discussion/Feedback collection in progress . So far we have got positive response from all the operators.**

- **May first week** : Presentation to Architecture committee – Seek feedback on problem statement and overall approach

- **May first week** : Presentation to MSB, Ext-API and identify areas where we can work together – Discussion with MSB completed , Discussion with Ext-API scheduled for Wednesday, 8th May
  - **MSB team thinks the proposed capability has some overlap with the features in roadmap that can be developed with additional plugins**

- **May last week, June :** Consolidate feedback and present to Architecture/TSC Committees for potential development in E or F Release

Proposed Use Cases (Any one to start with)

Scenario 1 :
Dynamic Routing and Request/Response Transformation for SOL005 API

- API GW Exposing two types of interfaces
  - **Simplified internal API which hides SOL005 API or API exposed by external NFVO**
  - **Pure SOL005 which can be used for integration with OSS/BSS**
- Use Case
  - **Case 1) ONAP Component wants to access an External NFVO for LCM operation (sub domain)**
  - **Case 2) ONAP Component wants to work with a component internal/external via SOL005 API**
- Operation
  - **Case 1: API GW takes care of transforming the simplified internal API to corresponding API calls to external NFVO APIs**
  - **Case 2 : API GW receives SOL005 API calls and enriches/transforms the API with internal/external API call**

Scenario 2 :
Dynamic Routing and Request/Response Transformation for SOL003 API

- API GW Exposing two types of interfaces
  - **Simplified internal API which hides SOL003/Vendor complexity**
  - **Pure SOL003 (without VNFM specific extensions)**
- Use Case
  - **Case 1) ONAP Component wants to use Simplified API for VNF instantiation**
  - **Case 2) ONAP Component supports pure SOL003 API but not aware of vendor extensions**
- Operation
  - **Case 1: API GW takes care of transforming simplified internal API to corresponding multiple API calls - SOL003 specific or Vendor specific APIs**
  - **Case 2: API GW receives pure SOL003 request and enriches the request with vendor specific SOL003 extended parameters**

Thank You

Netcracker
An NEC Company