Microservices Bus Proposal(5/12/17)

Project Name:

- Proposed name for the project: `Microservices Bus`
- Proposed name for the repository: ~~`msb.onap.org`~~ msb

Project Description:

Microservices Bus provide a reliable, resilient and scalable communication and governance infrastructure to support Microservice Architecture including service registration/discovery, service routing and load balancing, timeout and retry, requests tracing and metrics collecting, etc. It's a pluggable architecture so it can plugin service provider options like AAF to provide Authentication & Authorization for APIs. Microservices Platform also provides a service portal and service requests logging, tracing and monitoring mechanism, etc.

Source codes are from OPEN-O and have already been used to support Microservice Architecture of OPEN-O for the vEPC and VoLTE use cases in its two successful releases.

Note and Clarification:

- We have reached consensus with OOM project that MSB and OOM manage service registry at 2 different levels, MSB at the micro/macro service endpoint level vs OOM at the micro-service/component level, MSB will work with OOM to provide a comprehensive **Service Registry** function.

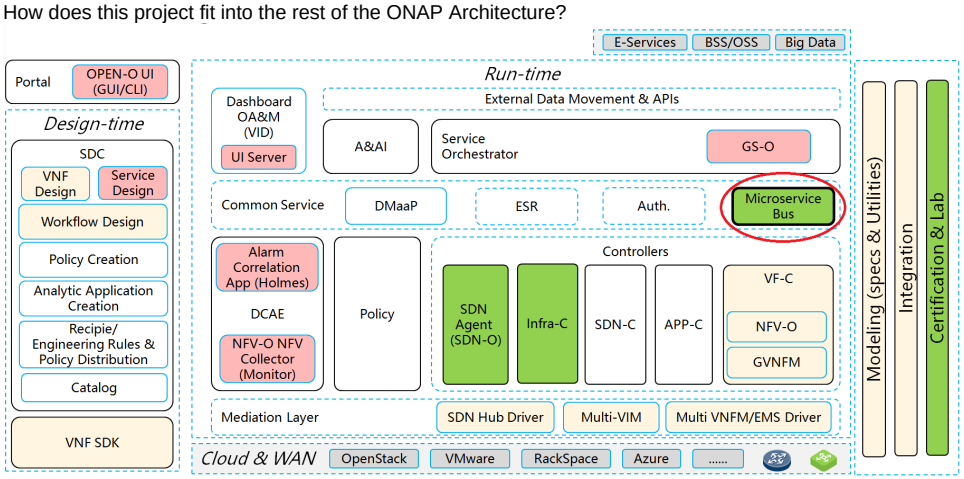
- MSB provides a common infrastructure for ONAP services, no matter it's a "micro" service or a "macro" service. In this context, the term "microservice" just means that the service's scope is highly organized around one business capability so it's smaller than "normal" service. Just like micrcoservice, other services wich is "bigger" is also a process which needs to expose service endpoint to be accessed by its consumers. So both "micro" and "normal" services need the service registration/discovery/routing/load balancing mechanism provided by MSB.

Scope:

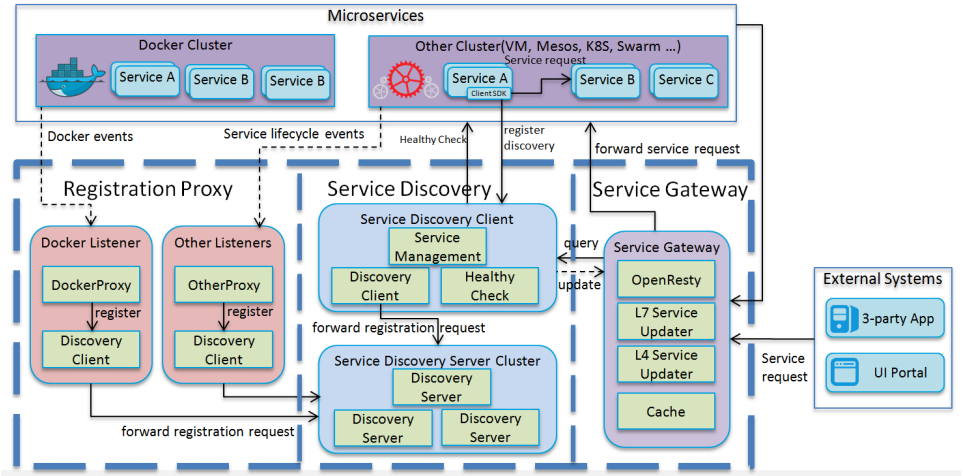
- Service registration
  - Registration via Restful API
  - Registration via portal
  - Registration via proxy
  - Note: Registration info is used for service request routing, the info including service name, service exposed url, version, service instance endpoint(IP:Port), service protocol, service ttl, service load balancing method, etc.
- Service discovery - Server side discovery
  - service request routing
  - service request load balancing
- Service discovery - Client side discovery
  - client side discovery SDK
- Service discovery - DNS
  - Discovery and load balancing by DNS server
  - Service consumer directly talk to service provider
- Service Health Check
  - Note: The goal of service health check of MSB is to maintain the correct health status of service at endpoint level in the service registry so the service consumer will not get a failed service provider instance, MSB doesn't try to kill and restart the onap component, which is the scope of OOM(ONAP Operations Manager).
- Service API gateway
  - Client request routing
  - Client request load balancing
  - Transformation, such as https to http
  - Provide authentication & authorization for service request with plugin of auth service provider like AAF
    Note: MSB itself doesn't provide auth service, which is provided by a auth service provider microservice such as AAF(Authentication and Authorization Framework)
  - Service request logging
  - Service Request Rate-limiting
  - Service monitoring
  - Request result cache
  - Solve cross-domain issue for web application
  - Other functionalities with the pluggable architecture capability ...
- Service API Portal
  - Provide a Service API Portal to expose all the ONAP Swagger format API descriptions
  - Don’t need to maintain an independent API Portal and API description documents – save money and time
  - Keep the consistency of the API document with the source code
  - Support multiple versions of APIs
  - Always get the latest API documents of the current development branch which is generated by CI/CD automatically
  - Provide self-service onboarding for developers
- Client SDK
  - Service registration&discover client SDK
  - Service request client SDK
- Service Mesh
  - MSB is planning to support Service Mesh in future release

Architecture Alignment:

- How does this project fit into the rest of the ONAP Architecture?



  - Please Include architecture diagram if possible



  - What other ONAP projects does this project depend on?
    - Integration
    - OOM
- How does this align with external standards/specifications?
  - APIs/Interfaces - OpenAPI/Swagger
  - Information/data models - Swagger JSON or YAML models
- **Are there dependencies with other open source projects?**
  - APIs/Interfaces - OpenResty, Consul, Redis, Istio
  - Integration Testing
  - etc.

This part was added yesterday and have not been discussed inside project yet, we will discuss it later and solve it inside project. Currently, this part is considered one of the potential seed code, but it doesn't impact the scope of this proposal.

****

DME2 (Direct Messaging Engine)

DME2 provides encapsulated JMS via HTTP transport APIs that a Service provider instance can use to dynamically register its end point on service provider startup and to unregister its end point when it shuts down. Service endpoints are persisted on the Cloud Data Store for the purpose of endpoint dynamic discovery as the client requests are being routed.

blocked URL

DME2 also provides client API's to route Service requests based on data, partner or geographic affinities.The requests are effectively load balanced and client requests can failover dynamically between the service endpoints.

DME2 Key Features

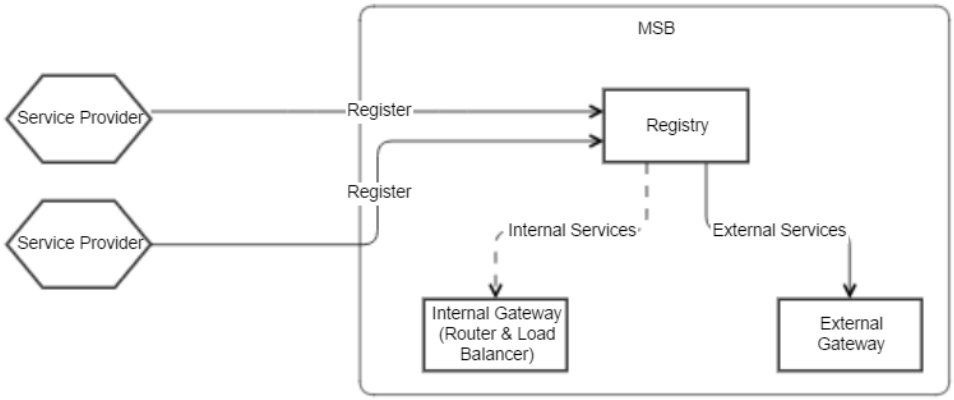
DME2 provides a robust set of features for any service provider implementation to utilize for inter-process client/server communication, including:

- **Fast-failover** - Queuing the requests on server implementation is disabled by default, which allows the client to fast failover if a service provider is consumed at its max capacity.
- **Dynamic Routing** - The client requests will be routed to the DME2 service provider instances via routing made possible by the dynamic registration of the service provider as instances become available
- **Affinity Routing** - The client requests will be routed to the service provider by means of data, partner or geographic affinity depending on the client requirements.
- **JMS API's** - Provides standard JMS API implementation for destinations type as JMS Queues or Point-to-Point messaging
- **Async Request/Reply** - Enables better use of resources by avoiding the traditional thread per request, thus supporting highly scalable service provider implementation

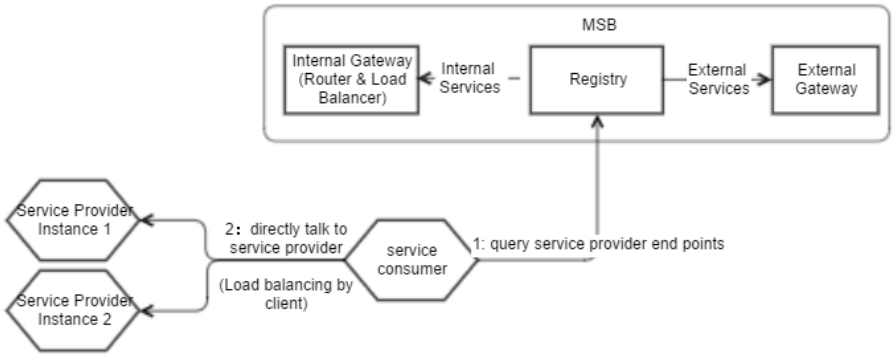
****

MSB Use Cases:

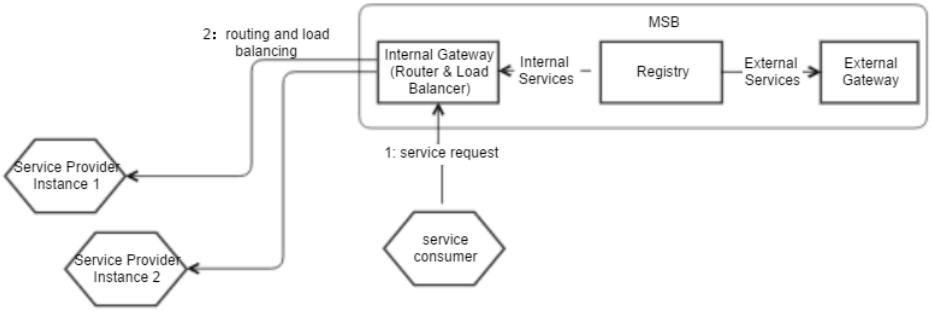
1. **Service registration per service provider instances**
   - **Registration**
     Service instances are registered to the registry by proxy or themselves. The visible scope should be indicated as a parameter when register. If a service is only internal visible, the service information is only pulled by the internal gateway (aka router & load balancer) and used by other components inside the system, the interal services can't be accessed by external systems or front end(web client). If a service is visible to external system, the service information is pulled by the external gateway and can be accessed by external systems and front end (web client) with auth.



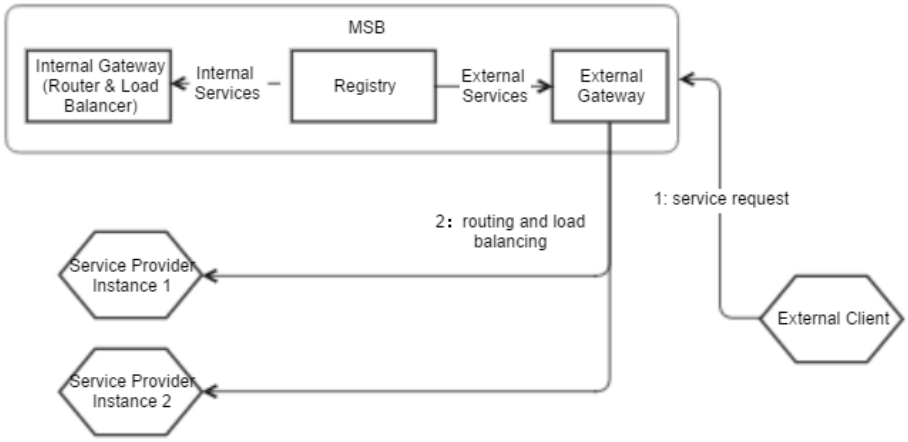
   - **Discovery & Service Consuming**
     - For internal service consumers(Components inside ONAP system, such as A&AI, SO, Controller, etc.)
       - Client side discovery and load balancing



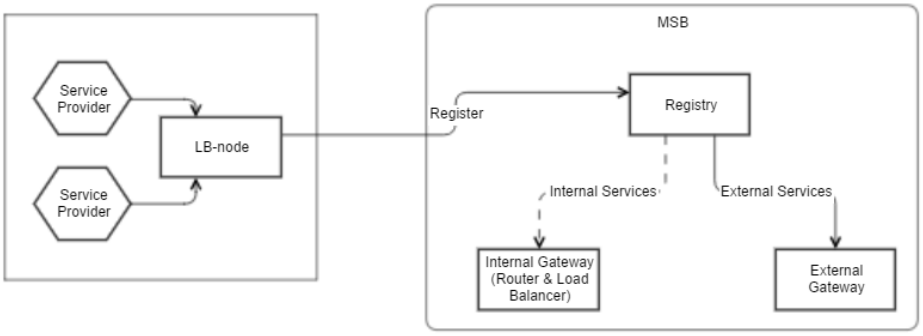
       - Server side discovery and load balancing



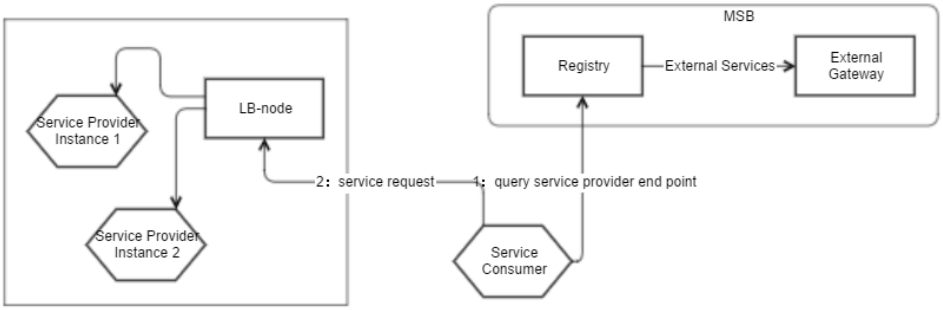
- For external service clients(OSS, BSS, Web client, etc.), access the service via external gateway



2. **Service registration per service**
   The service may have its own load balancer built inside, for example, Kubernetes can create a load balancer for a service. In such case, only need to register the service LB node to MSB, and the service request from the consumer is routed to the service LB node.
   - **Registration**

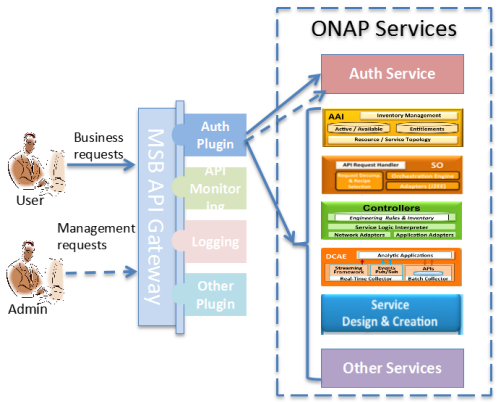


   - **Discovery & Service Consuming**



   Note: Only show the client side discovery in this diagram for simplicity, it's also possible to use server side discovery by the internal gateway.

3. **Centralized Authentication&Authorization via MSB plugin**
   MSB is a pluggable architecture, so it can provide centralized authentication & authorization for service request with plugin of auth service provider like AAF.



Centralized Authentication
1. User send a service request to MSB API Gateway
2. MSB API Gateway auth plugin check the auth token
   2.1 If a valid token exist, MSB API Gateway forward the request to the destination service provider
   2.2 If not, MSB API Gateway forward the request to the Auth Service, and redirect user request to login page
   2.3 Auth service create a token cookie after user login with valid name and password

Centralized Authorization(Assuming user already login)
1. User send a service request to MSB API Gateway
2. MSB API Gateway auth plugin send the user token and request(Http method + Resource url) to Auth Service
   2.1 If user has the permission, MSB API Gateway forward the request to the destination service provider
   2.2 If not, MSB return operation not allowed error to user

Centralized User, Role and Permission Management
Centralized in the Auth Service

Note: Auth Service is not in the scope of MSB

Resources:

- Primary Contact RamKoya, HuabingZhao, Al Hua, Sanjay Agraharam, Brijesh Khandelwal
- Names, gerritIDs, and company affiliations of the committer

Name	Gerrit ID	Company	Email	Time Zone
Ram Koya		AT&T	rk541m@att.com	Dallas, USA. CST/CDT
John Murray		AT&T	jfm@research.att.com	Bedminster, USA EST/EDT
Dominic Lunanuova		AT&T	dgl@research.att.com	MiddleTown, USA EST/EDT
Chengli Wang		CMCC	wangchengli@chinamobile.com	Beijing, China. UTC +8
Tao Shen		CMCC	shentao@chinamobile.com	Beijing, China. UTC +8
Zhaoxing Meng	Zhaoxing	ZTE	meng.zhaoxing1@zte.com.cn	Beijing, China. UTC +8
Huabing Zhao	HuabingZhao	ZTE	zhao.huabing@zte.com.cn	Beijing, China. UTC +8
Yonggang Wang		ZTE	wang.yonggang131@zte.com.cn	Beijing, China. UTC +8
Tang Hua		ZTE	tang.hua52@zte.com.cn	Beijing, China. UTC +8
Brijesh Khandelwal		Tech Mahindra	kbrijesh@TechMahindra.com	CDT, USA.
Hailong He		ZTE	he.hailong5@zte.com.cn	Beijing, China. UTC +8
Jian Ming	jianming	ZTE	jian.ming@zte.com.cn	Beijing, China. UTC +8
Leibo Huang		ZTE	huang.leibo20@zte.com.cn	Beijing, China. UTC +8
Hu Rui		ZTE	hu.rui2@zte.com.cn	Beijing, China. UTC +8
Jinquan Ni		ZTE	ni.jinquan@zte.com.cn	Beijing, China. UTC +8
Bala		IBM	balasman@in.ibm.com	Middletown, NJ. EDT/EST

- Names and affiliations of any other contributors

Name	Gerrit ID	Company	Email	Time Zone
Jochen Kappel		IBM	jochen.kappel@de.ibm.com	

- Project Roles (include RACI chart, if applicable)

Other Information:

- link to seed code
  https://gerrit.open-o.org/r/gitweb?p=common-services-microservice-bus.git;a=tree;h=refs/heads/master;hb=refs/heads/master
  git clone https://gerrit.open-o.org/r/common-services-microservice-bus
  https://github.com/att/dme

- link to API documentation
  Microservice Bus API Documentation
  https://github.com/att/DME/blob/master/Readme.md
- Vendor Neutral
  All proprietary trademarks, logos, product names, etc. have been removed.
- Meets Board policy (including IPR)
  Yes

Use the above information to create a key project facts section on your project page

Key Project Facts

Project Name:

- JIRA project name: Microservices Bus
- JIRA project prefix: MSB

Repo name:
Lifecycle State: Incubation
Primary Contact: RamKoya, HuabingZhao, Al Hua, Sanjay Agraharam, Brijesh Khandelwal
Project Lead: Huabing Zhao zhao.huabing@zte.com.cn
mailing list tag [msb]
Committers:

please refer to the above table

*Link to TSC approval:
Link to approval of additional submitters: