Modularity – Proposal for achieving functional decomposition

Manoj Nair, NetCracker, May 2018
(with input from Alex Vul, Parviz Yegani, Nigel Davis)
Modularity

How functional capabilities can be decomposed and regrouped for flexible deployments, to create functional variations.

How to decompose?
- Based on business/domain capabilities (Domain Driven Design)
- Based on Verb – Example Collect, Analyze, Configure, Decompose
- Based on Noun/Entities – Example Service, VNF, Policy
- Based on software layers – Example DB layer, UI, Backend

Why we need modularity?
- Standardize capabilities
- Have optional alternatives
- Enable functional variations
- Enable flexible deployment patterns
- Enable model driven capabilities

Microservice Architecture is a natural enabler for modularity
- Independent development and testing
- Enable CI/CD/CT

Followed in ONAP
Modularity, Model Driven, Cloud Native – Some misconception

- Modularity ≠ Microservice
  - Boundary, Style of Decomposition matters
- Model Driven – Model driven for intent or Model Driven for Solution behavior?
  - Intent: End state to be achieved, represented through IM/DM.
  - Behavior: Behavior of solution to be tuned to achieve an end state represented through configuration model
    - Configuration of modules – Implementation meta model
    - Sequence of actions – Workflow
- Cloud Native ≠ Docker, ≠ K8S, ≠ Cloud based
  - Applications with built-in capability to leverage the agility and distributed nature of Cloud environment
How Modularity achieved today in ONAP? What is Microservice in ONAP?

Different views on Microservice

- Docker Container
- POD
- Kubernetes Service
- Kubernetes Deployment
- Component
- A Functional Entity
- API End Point
- Application
- A Feature of an application

- ONAP Follows a Project Specific view of Modularity
- Each project follows own methodology for decomposition of functionality
- Most of the projects follow a software centric functional decomposition rather than domain capability decomposition

- These are all different levels of functionality or characteristic ONAP stakeholders use at different contexts.
- Two aspects to be considered – what is the level of granularity of the functionality, what is the context (deployment, information, runtime etc)

ONAP modularity is currently driven by projects with software module (DB, Tools etc) based boundary, not domain capability based boundaries
Problem Statement 2: Modularity and Tight Dependency

- Heavy fulfilment and assurance stack with tight dependency between components
- Heavy dependency on the design time environment for enabling capabilities
- Each module follows own style and category of APIs and defines own set of entities requiring dependent modules to evolve continuously
- Business logic and internal APIs to access the business logic tightly coupled.
- Operational stacks are not easily replaceable.
- Stack components cannot be easily replaced due to tight dependency
Problem Statement 3: Top-down vs Bottom-up Approach

Balancing these two approaches is necessary for production deployment of ONAP
How modular functional capabilities that are aligned to standard business processes help in top-down operational transformation
How ONAP modules are decomposed today? (Reference Amsterdam Microservices)

Not functional decomposition, but software decomposition, decomposition logic based on discretion of project owner.

DCAE is a software platform to host domain apps. More of a task based decomposition (collection, analysis, storage), mix of platform and domain services.
Modularity: How Microservice Composition can be Represented? Learnings from TMForum TAM

What we can learn from TAM:
- Define Microservice Levels (Level 1, Level 2 etc) that compose domain functions
- Similar (need not be same) in concept to TMForum TAM with different levels of application capability decomposition.
- Gives option to select the application capabilities based on the well known TAM like capability decomposition map.
- End user can select the capability and associated micro services (at any level) enabling those capabilities.
- Help identify functions and associated stakeholders in ONAP at different levels of granularity

Benefits:
- Consistency in Functional decomposition
- Single terminology across layers and across projects
- Close relationship with Business process terminology and business process flows
- Can reference APIs (Open APIs) rather than proprietary APIs
A component-system pattern is fractal, split a component and you get more components, combine components and you get a component.

MicroServices can be perceived as a component fulfilling a cohesive function.

A Branded Element constitutes of multiple features which in turn are realized through set of functions – all levels can be a composite component.

Solution

A Branded Higher Level Component

Feature

Function

Coherent and useful unit of functionality

All different levels of microservice – Macroservice, Miniservice, Microservice

Reference: TMForum IG1118
ONAP Capability Decomposition vs TAM Levels
(Example – Not Complete)

ONAP Components
- Orchestrator
- Generic Controller
- A&AI
- SDC
- Multi-VIM
- DMAPP
- MSB
- AAF
- DCAE
- SDNC
- Policy
- External API

ONAP Microservices with core domain logic

TAM Level 1
- Service Capability Orchestration
- Service Inventory Management
- Service Order Management
- Resource Catalog Management
- Resource Inventory Management
- Resource Capability Orchestration
- Resource Fault Management
- Resource Performance Management
- Resource Quality Management
- Resource Assurance Control

TAM Level 2
- Service Activation Management
- Service Inventory Reconciliation
- Service Order Orchestration
- Service Design and Assign
- Service Configuration Management
- Service Resource Reconciliation
- Service Workflow Control
- Service Performance Monitoring
- Service Instantiation and LCM
- Service Workflow Control
- Service Quality Collection & Monitoring
- Service Quality Reporting
- Service Quality Model Establishment
- Service Workflow Decomposition
- Service Performance Reporting
- Service Onboarding
- Resource Inventory Reporting
- Resource Inventory Update
- Resource Inventory Retrieval
- Resource Design and Assign
- Resource Order Orchestration
- Resource Fault & Performance Data Mediation
- Resource Discovery
- Resource Activation
- Resource Commissioning & CM

THE LINUX FOUNDATION
Modularity through Microservices

A functional capability expressed in multiple levels of granularity. Each granular level is independently deployable, expose well defined APIs. Granularity level left to the user - what independent capability mix is required at what level.

Level 1
- Microservices composing a business facing capability that is traditionally having end to end operational scope and technology neutral.
- Typically exposes Standard End to End Customer Facing Service Level API interfaces –TMF 641 etc.
- E.g. Service Order Management, SQM, SLAM, Service Catalog Management, Service Capability Orchestration, Resource Inventory Management, Resource Order Management, etc.

Level 2
- A coarse-grained micro service logically grouping a set of features as branded component that can be exposed as an independent service with well defined APIs.
- Typically exposes a standard API with scope limited to the functions being composed into the logical group
- E.g – NFVO, VNFM, VIM, EMS, Generic Controller, SDN Controller

Level 3
- A micro service realizing a single independent feature associated with a high level capability or platform capability
- Typically expose APIs associated with a single feature. Internally route the APIs to independent functional microservice
- E.g. - For Service Capability Orchestration: Decomposition, Onboarding, Instantiation & LCM, Workflow Management

Level 4
- A logically independent and tightly scoped component. Independently deployable and scalable.
- Typically exposes APIs between functions with internal scope
- E.g. Workflow engine, Runtime catalogue, VIM adaptor

Level 0 is assumed to be a Solution which is composed of Level 1 MS – E.g. Domain Orchestrator
Modularity through Microservices: Personas at each level

Level 1
- Microservices composing a business facing capability that is traditionally having end to end operational scope and technology neutral.
- BSS User, Operator Operations Staff, Business User, BSS Partner, Integration Engineer

Level 2
- A coarse-grained micro service logically grouping a set of features as branded component that can be exposed as an independent service with well defined APIs.
- Operations Staff, DevOps Engineer, End to End Tester, Use Case Implementer, Integration Engineer

Level 3
- A micro service realizing a single independent feature associated with a high level capability or platform capability
- Feature Tester, Use Case Developer, ONAP Project Developer, DevOps Engineer

Level 4
- A logically independent and tightly scoped component. Independently deployable and scalable
- ONAP Project Developer, DevOps Engineer

Level 0 is assumed to be a Solution which is composed of Level 1 MS – E.g. Domain Orchestrator
Modularity – Functional Layering (Only for representation, not complete)

Level 1
- Service Order Management
- Service Inventory Management
- Service Lifecycle Management
- Service Capability Orchestration
- Service Test Management
- Service Performance Management
- Service Problem Management
- Service Assurance Control
- Resource Catalog Management
- Resource Lifecycle Management
- Resource Test Management
- Resource Order Management
- Resource Performance Management
- Resource Capability Orchestration
- Resource Fault Management
- Resource Domain Management
- Resource Usage Management

Level 2
- Orchestrator
- VNF Manager
- VIM
- SDN Controller
- EMS
- SDC
- NMS
- DCAE
- Multi-VIM
- Policy
- A&AI
- Ext-API

Level 3
- Service Design and Assign
- Service Order Orchestration
- Service Configuration Management
- Service Activation Management
- Service Inventory Reconciliation
- Service-Resource Inventory
- Service Instantiation and LCM
- Service Decomposition
- Service Onboarding
- Service Workflow Configuration
- Service Workflow Control
- Service Performance Monitoring
- Resource Commissioning & CM
- Service Performance Analysis
- Service Performance Reporting
- Service Quality Model Establishment
- Service Quality Collection & Monitoring
- Service Quality Analysis
- Service Quality Reporting
- Resource Inventory Reconciliation

Level 4
- DMaaP Message Router
- DMaaP Data Router
- MSB Registration Proxy
- MSB Service Discovery Cluster
- SDC Persistent Store
- Orchestration Workflow Engine
- Consul Cluster
- Common Controller SDK
- Kubernetes
- Distributed Data Store
- Multi-VIM Plugin
- VIM Adaptor
Alternate Layering/Grouping of Microservices for Modularity

Domain Application Components (Kubernetes Service)

- Ext-API
- SO
- A&AI
- Policy
- SDC
- Generic Controller
- SDNC
- DCAE

Domain Services (Kubernetes Deployment)

- MSO
- A&AI Service
- SDC Catalog API Gateway
- DCAE Deployment Handler
- SDC Distribution Engine
- DCAE Service Change Handler
- A&AI Traversal
- DCAE Policy Handler
- A&AI Resources
- App-C Business Logic
- Portal Apps
- Vfc-catalog
- vfc-nslcm
- Vfc-workflow
- VFC-resmgr
- vfc-vnfmgr

Common Services (Kubernetes Deployments)

- SDC Catalog
- Logging
- DMaaP
- MSB
- Configuration Data Store
- MUSIC
- Common Controller SDK
- OOF
- Multi-VIM Plugin
- Data Collector
- PAP
- AAF
- CLAMP
- DCAE Controller
- PDP

Platform Services (Kubernetes Deployment)

- Consul
- Kafka
- ELK Stack
- CDAP
- Multi-VIM Plugin
- Workflow Engine
- VIM Adaptor
- DGBuilder
- Zookeeper

Expose External APIs

Expose Domain Functionality

Internal API

CNCF tools play a role here

Level 2

Level 3

Level 4
ONAP Capabilities mapped to TAM (Example 1)
ONAP Capabilities mapped to TAM (Example 2)
Mapping of Microservice Levels to ONAP (Example - Not Complete – As-Is)

Level 1

ONAP Microservices realizing an independent feature or group of features that are mapped to a standard component

Level 2

DCAE
SDNC/App-P-C
Orchestrator
VFC
A&AI
SDC
Multi-VIM
Policy
External API
DMAPP
MSB
AAF
NFVO
VNFM

Level 3

Data Collection Service
Service Deployment
TCA
API Gateway
MSO
A&AI Service
A&AI Distribution Engine
A&AI Catalog Logic
PAP
PDP-X
PDP-D

Level 4

VES Collector
Controller
Controller Platform
Runtime Catalog
Workflow Engine
Dynamic inventory Graph
Message Router
Cassandra
Policy Config Repo
BRMS GW
Oper Policy Store
Rules Engine
Controller Platform
CDAP Analytics Platform
Workflow Engine
Dynamic inventory Graph
Message Router
Cassandra
Policy Config Repo
BRMS GW
Oper Policy Store
Rules Engine

Identify functional levels of existing microservice

ONAP Microservices which are supporting domain functionality

ONAP Microservices which are independent and providing platform functionality
Tiger team feedback (Alex, Parviz, David, Nigel)

- **API Alignment – External and Internal API**
  - This may be the first step in decomposing domain functionality without making any major structural change of projects

- **Service Mesh and CNCF Project (Parviz’s Slide deck) Integration**
  - Increased interest in CNCF toolset, how a Service mesh based “intelligent edge- dump pipe” model can be achieved

- **Statelessness of Microservice**

- **Transaction support across microservices (Do we really need this?)**

- **What changes required in CI/CD/CT to support Modularity – Automated Integration Tests**, modular deployment composition.
Microservice Functional Decomposition – Example DCAE

Operational Capabilities

- Service Problem Management
- Service Quality Management
- Service Assurance Control
- Service Performance Management
- Resource Performance Management
- Resource Fault Management

Level 1

Branded Component

Façade enabled through MSB (Expose Operational API via External API)

SDC
CLAMP
DCAE

Façade enabled through MSB, Service Mesh Fabric (Exposé Internal API supporting group of entities of External API)

Assurance Service Design & Deployment
Service Change Handler
Deployment Handler
Inventory

CBS

Feature

Level 2

Monitoring
Holmes

Analysis
TCA

Reporting
CLAMP Dashboard

Correction and Resolution
Policy Handler

DMaaS
CDAP Broker
CDAP Cluster

VES Collector
SNMP Collector

DMaaS
CDAP Broker
CDAP Cluster

Level 3

Data Collection

Level 4

Monitoring

Assurance Service Design & Deployment

Grouping of capabilities as required by the operational end user and business process modeler

NOT IMPLEMENTED IN ONAP

CLAMP Desk

Elk Stack

CDAP Broker

CDAP Cluster

PGaaS

Cloudify

Policy Handler
Functional Decomposition – Observation

- Operational capabilities currently supported in ONAP are not easily decomposable – vertically or horizontally - as there is tight coupling between different microservices
- This will limit an operator’s preference for top down approach of operational capability enablement
- Need to identify and logically separate (at least at the API level) domain specific capabilities that can be easily identified for defining business process flows
Information Context of Microservices – Key Considerations

• In ONAP there are different types of models used to control different aspect of Microservices
  - behavioral aspect – To represent intended behavior of Microservice which are represented as configurations to achieve model driven capabilities – this may be required to control MS behavior at runtime
  - end result aspect/Intent – To represent end state expected to be achieved as per business and customer demand – This may be an input to microservices to carry out various actions
  - deployment aspect – To represent how microservices are deployed and functionalities are realized – For various deployment options

• All these aspects can be mapped to microservice layers
Information Context – Behavioral Aspect - Example DCAE - (To be verified by OOM & DCAE)

Service template TOSCA for CL (CLAMP) - DCAE CL Blueprint

- Configuration Policy
- Operational Policy
- Closed Loop Configuration
- Policy Recipes for SO

DB Schema (Inventory) - VES Schema - DCAE Component Metadata - EMF

- DB Schema (Policy Config)
- XACML
- Drools Rules
- PIP Schema

Representation of what model elements used to configure DCAE, CLAMP and Policy to achieve end result

Information used by DCAE, CLAMP to provision dependent Components using internal API

Low level schema used by platform level micro services
Information Context – Intent – Example DCAE - (To be verified by OOM & DCAE)

Example SID ABEs

Representation of Intent

Information used by DCAE, CLAMP to provision dependent Components using internal API, Information managed by Level 3 MS

Low level schema used by platform level micro services
Deployment Aspect – Deployment Configuration for each Microservice Layers (To be verified by OOM & DCAE) – Example DCAE

Level 1
Deploys a custom ONAP solution depending on the domain functionality required

Level 2
Deployment Context of Coarse domain functionality as Service

Level 3
Deployment Context of Aggregate Features as Deployments

Level 4
Deployment Context of Micro functionalities as Pods/Docker Containers
Observation on Information Context

• No clear differentiation between behavioral aspect (configuration of MS), and intent (what to be achieved – create CL).

• Majority of entities managed by Microservices are those to deal with platform capabilities

• Not all domain capabilities are available as managed entities
Operational Context 1/2

• Day 0:
  - Receives service order for enabling end to end service
  - Wait for notification from BSS on availability of partner services/resources

• Day 1:
  - Query ONAP Service catalog for the infrastructure service corresponding to the partner domain (Not done currently)
  - Send an activation request to partner domain for infrastructure service (Not done currently)
  - Place a service instantiation request on ONAP over ETSI Os-Ma
  - Carry out end to end testing (Not done currently)

• Day 2:
  - Send the customer configuration in terms of FW, QoS, performance monitoring configurations to partner SDNC over TAPI interface

• Day 3:
  - Tune partner SDNC to meet end to end SLA.
Operational Context – 2/2 – DCAE

Day 0 (Design and Order)
- Service Design/Assign
- Service Order Orchestration
- Service Capability Orchestration
- Ext-API (For Assurance Service Order Item from OSS)
- SDC
- CLAMP
- Service Change Handler
- CBS
- Inventory
- Consul Cluster
- Cloudify
- PGaaS

Day 1 (Service and Resource Provisioning)
- Service LCM
- Service Configuration & Activation Management
- Resource Discovery & Activation
- CLAMP
- DCAE
- Policy
- Deployment Handler
- Inventory
- PAP
- Service Change Handler
- PDP
- DMaaS
- CDAP Broker
- Drools Engine
- Cloudify
- VES Collector
- Policy Handler

Day 2 (Customer Configuration)
- Resource CM
- Resource Discovery & Activation
- Resource CM
- CLAMP
- APP-C (for VES Agent Config)
- Policy
- VES Collector
- TCA
- PDP
- PAP
- TCA
- VES Collector
- PDP
- PAP

Day 3 SLAM
- SLAM & SQM
- Level 1 Business facing capabilities
- DCAE
- Policy
- CLAMP
- Level 2 Operational & Designer User facing Micro Services
- VES Collector
- TCA
- PDP
- PAP
- VES Collector
- TCA
- PDP
- PAP

- Level 3 ONAP managed service team facing capabilities
- VES Collector
- TCA
- PDP
- PAP

- Level 4 ONAP Engineering/Integration team facing capabilities
- Policy Handler
- DMaaS
- ELK Stack
- PGaaS
- CDAP Broker
- CDAP Cluster
- Cloudify
- Consul Cluster
- Consul Cluster
- Cloudify
- DCAE
- CLAMP
- SDC
- CLAMP

Observation on Operational Context

• Functional overlap between Day 0 to Day 3 Operational Functions
• From Managed Services point of view it will be difficult to segregate the stages of operations as same functions repeat across operational stages.
• Level 4 support teams for Operations (Typically engineering) need to be expertise across different branded components to provide effective support
Currently none of the Assurance capabilities are exposed through Ext-API

Level 1

Facade enabled through MSB (Expose Operational API via External API)

Level 2

Facade enabled through MSB (Expose Internal API supporting group of entities of External API)

Level 3

Facade enabled through MSB at platform level

Alex: Façade for microservices – Focus on the functional APIs and expose those from microservices
API Gateway and Service Mesh

Most API Gateways have features offered from Service Mesh built in. However, they can still leverage existing service mesh too.

API Gateway expose (micro) services as managed APIs

Service Mesh decouple and offload most of the service-to-service communication from business logic.

Service mesh is merely an inter-service communication infrastructure which doesn't have any business notion. So it will be ideal to be used at lower levels of Microservices.
External API along with MSB is expected to play the role of API mediation between External and Internal API. Alternatively Service Mesh can be implemented at Level 3 or Level 4 to support API mapping and routing functionality.

External APIs
TMF, ETSI, MEF, ONF APIs Alignment
Good to have API Gateway here, Service Mesh is optional

Support specific entities associated with TMF, ETSI, MEF, ONF APIs
AND/OR
ONAP Internal Feature Level APIs
Good to have Service Mesh here
State Handling of Microservice – Example DCAE – Current

Potential DCAE functions that maintain states:

- Healthcheck, reachability, XNF State
- Alarm correlation, Rule processing state
- Closed loop processing state

To be confirmed with DCAE team!
DCAE n

Distributed Configuration Store (OOM)

- Music has distributed locking mechanism to control processing across different instances.
- OOM already maintains a Consul cluster. This can be reused for DCAE without a dedicated cluster.
States - Distributed concerns – What CAP Model ONAP Components Require

- CAP – Consistency, Availability, Partition Tolerance
- CAP Theorem – Only any two properties in CAP can be supported. The properties are selected based on business requirement.
- Saga Pattern or State at Source: Each local transaction within a microservice (that is having own DB to manage state) triggers an event and the dependent services updates the local copy of the state.
- Have common services to support different consistency models across microservices
  - Strong consistency: All provisioning/activation/configuration (mastership), workflow management function
  - Eventual consistency: Monitoring data, Inventory, Topology
- Music supports an eventually consistent store
- Need to identify Microservice functions which require a strong consistency mechanism
- Consistency mechanisms
  - Raft
  - Gossip
- Tools for distributed consistency management
  - Zookeeper
  - Hazlecast
  - Redis
Transaction Support – How MS enable ACID properties

- ACID – Atomicity, Consistency, Isolation, Durability
- Potential areas in ONAP currently using Transaction Model
  - SO interaction with SDNC, VIM driven by workflows (need error mitigation workflows to support atomicity)
  - Closed loop control flow – Need to have separate event handling mechanism to revert any false positive trigger conditions
  - Distributed transactions in an Active-Active HA deployment
- It is a good practice to avoid transaction across microservices especially in a distributed environment
- Use Saga Pattern (described in previous slide). But with caution- it can lead to complexity, cyclic dependency.
- One option for avoiding erroneous transactions is to use two phase commit
- In distributed transactions, to avoid race conditions it is a good practice to use distributed lock with the key assigned to a master
Microservices – Building blocks (Tools to be developed to ease development, ensure consistency)

• Potential Options
  - Microservices SDK: Violates independent development concerns, tooling requirement
  - Code generation: Possible – Meta language to be chosen
  - Configuration Templates – Selection of template language, compatibility with different development styles
  - Enforce guidelines: Require tooling – Sonar for example with additional rules

• Suggested Approach
  - Each microservice owner (depending on the layer of functionality) to define APIs as per a predefined guideline – Potential option Swagger
  - Generate code based on Swagger API, MS owner to implement the APIs and register API end point to configuration templates through meta information
  - Configuration templates: Define standard configuration template for MS meta information and dependencies
  - OOM, MSB to read the configuration templates to do wiring between MS
Next Steps
Way forward – Summary

**Short term:**
- ONAP microservices identification from top down functionality point of view rather than project specific microservice grouping
- Regrouping of microservices to segregate microservices providing domain functionality, common microservices and platform microservices
- API layer separation from business logic
- Domain capability enablement through APIs at different levels of micro functionality
- Architecture/Design review to verify the microservices alignment to overall ONAP level goals rather than limiting to project goals
- Project teams to come up with strategy to separate domain functions and platform functions
- Stateless and Stateful MS – Strong and Eventual Consistency Mechanism Requirements – Leverage Distributed/shared storage if possible.
- Checklist to verify MS readiness across projects

**Long term:**
- Define microservices based on domain functionality, i.e. common functionality to be regrouped to a lower level microservice
- Create a microservice map based on domain functionality which can be used as a capability catalog for operators to pick and choose the specific functionality required.
Initial Recommendations to project teams

All projects identify the functional layers of microservice – core business logic, common service dependency, platform functionality.

Optional: Refactoring to enable functional decomposition based on domain capabilities rather than software implementation layers.

API layer separation from Business logic.

Identify Consistency Model at Project and Microservice Level – Strong or Eventual consistency.

Identify Transaction Model across Microservices – Strategies for eliminating cross MS Transactions.

Service Mesh for Internal communication and API GW for External communication act like Façade to express domain functionality.

Projects to leverage common platform level MS reduce project level footprint.
Governance Model & Guidelines

Governance

- MSA compatibility and maturity assessment based on modularity, model driven, cloud native capabilities – Check list preparation
- Recommendations to project teams based on maturity assessment
- Tools recommendations and CI/CD recommendations to verify the MSA alignment (SonarQube updates)
- Checklist and measurability of MSA compliance on continuous basis
Governance - MSA Enforcement through Objective Measurement

• Checklist to be shared across projects in regular cycles
• Checklist to cover following things to evaluate a project’s compliance to MSA
  - Modularity
  - Model Driven Capability Enablement
  - Cloud Native Behavior
• A weighted score of the checklist response to be used as input for certifying projects for MSA Compliance
Guidelines for Projects
Recommendations - Microservice Structural Changes

**Short Term**
- Single responsibility principle – Try not to club multiple functionalities in to single microservices (for example SO)
- Have the API layer separated from core domain logic implemented by Microservice so that APIs can evolve independently of domain logic
- Identify stateful services and check the possibility of leveraging DBaaS from OOM or Music
- Enhance the APIs to have functional capability – for example define APIs to support top down approach, which will support top level service capabilities in terms of managing specific entities.
- Enable swagger based tooling for consistent API representation and code generation across MS
- Expand the Microservices API scope to cover Operational, Security and Functional capabilities
- Identify the domain entities managed by each microservice – domain entities are data elements that represent a domain object. Domain entities are accessed through the APIs.
- Identify the categorization of microservice – i.e those providing platform capabilities (for example those wrapping specific reusable tools), domain capabilities (i.e those implementing core business logic – for example service decomposition), and shared capabilities (those supporting the domain capabilities like VIM Adaptors)
- Reduce cross dependency across microservices through bounded context principles – i.e a domain entity identity changes across the microservice boundaries, let the dependent microservice manage the identities.
- Use consistent configuration model for microservices – leverage OOM provided Consul if possible
- Each microservice should be built with fault tolerance capabilities – i.e through high availability enabled through OOM, with capability to regain state after a failure and minimum impact to dependent services. This can be done through replicas in OOM, but state management across instances should be addressed.
- Follow the Microservices functional decomposition pattern suggested in this ppt – Branded Component, Feature, Function
- Allow versioning of microservices – from deployment perspective. Identify the impact of two versions running simultaneously
- Enable portability of Microservices – i.e. Microservice implementation should separate the platform dependent libraries to a common platform layer microservice for easy portability

**Long Term**
- Enabling OOM to provide a catalog of available services from which associated capabilities can be that can be deployed
Recommendations – Microservice Interaction

• Short term
  - Reduce chatty interaction between microservices. Design microservices with low coupling and high cohesion – i.e. reduce inter dependency between microservices and related logic is kept in single microservice
  - Separation of core business logic and intra microservice interaction mechanism – this can be an enabler to support service mesh in the mid term.
  - Avoid all hard wiring between components – i.e do not define dependencies at the code level instead define it through a metadata – this is applicable for API access as well. Example – SDC hard wiring for catalog access (from VFC)
  - For replicated microservices, leverage OOM provided HAProxy, Loadbalancer mechanism or leverage MSB provided load balancer
  - Distributed state management using eventually consistent data store rather than incorporating dedicated logic in microservices
  - Enhancement of Music to generate data change notifications which can be propagated through DMaaS
  - All interaction between microservices to be policy driven with capability to control the interactions from a central service – leverage MSB capabilities and Configuration Policies. Additionally, control interaction between microservices as per meta file that can be enforced by OOM during instantiation of MS.
  - Enable tracing of interaction across microservices – leverage CNCF tools such as OpenTracing. Incorporate this capability as part of Logging
  - Reduce interaction between microservices within a branded component through regrouping of internal dependencies and use Asynchronous communication pattern for all internal communication to avoid long delays. Use REST/HTTP at the external facing MS and use Asynchronous communication (pub/sub) if possible for internal communication. Wherever REST is used support notification mechanism for delayed response. All action invocations between MS is preferred to be asynchronous.
  - Distributed transaction, State management : Enable Saga or State at source pattern – especially for those services that are not using distributed data store. Saga is a sequence of local transactions where each transaction updates data within a single service, first transaction is initiated by an external request then each subsequent step is triggered by the completion of the previous one which is notified by an event.

• Long Term
  - Move towards event driven microservices – i.e state stored in a distributed eventually consistent data store and any change in state triggers a notification to subscribed microservices which act upon change in state
Recommendations – Microservice Operational Changes

• Short Term
  - Microservices interdependency to be represented in a meta file for the OOM system to bring up MS in sequence
  - Microservices capabilities and associated API end points to be represented through metafile for OOM to register the capabilities with MSB. Currently component level end points are registered, instead functional capability driven endpoint registration to be supported
  - Configurations of microservices to be represented in a consistent manner and not hard coding should be encouraged
  - All point to point interaction to be blocked and all interaction to be driven through MSB, API Gateway or DMaaP which are controlled through access policies
  - Follow the layering pattern suggested in this presentation for mapping Services, Deployments, Pods and containers
  - Project teams to understand the Pod and container concept used by K8S – all related cohesive functionality to be grouped under a Pod. Currently each container is mapped to a Pod. Project teams to regroup their functionality that need to fit into a Pod
  - Project teams to identify the impact of multi-tenancy in ONAP – i.e ONAP deployment shared by multiple operational tenants. This will lead to different instances of MS Pods to be deployed in the cluster. Identify the functionality that can be shared across tenants. Follow the platform, common functionality, business logic level grouping for low impact to support tenancy
  - All the inter MS interactions to be controlled through RBAC and associated policies.
  - OOM : Enable selective installation of capabilities by having a catalog of capabilities to choose from which maps to associated microservices deployments – dependency resolution to be done automatically
  - OOM: Support different flavors of deployment – x node cluster, single machine, distributed clusters etc.
  - OAM dashboard for Microservices – with potential Prometheus integration
Recommendations – Enablers for MSA – Tools, Techniques

- Swagger metadata and associated code generation for consistent representation of APIs
- Microservices health monitoring
  - Prometheus
  - Log tracing using CNCF toolsets
- Cloud native tool sets
  - Refer to a separate analysis done by Tiger team
- Template tools (Data mapping)
  - For model mapping from one format to other – Jinja2, Velocity etc.
- SDK tools
  - Eclipse custom plugin developed for microservices
  - Maven archetype for microservices project structure with DMaaP client, MSB client, Consul based configuration template etc.
- Tools for consistency and state Replications
  - MUSIC
  - Zookeeper
  - Redis
  - Hazelcast
- Sidecar and service mesh
  - Refer to separate analysis done by Tiger team
Thank You
Trouble to Resolve: TMForum recommended decomposition

**Service Trouble Management**

- Close Service Trouble Report
- Correct & Resolve Service Problem
- Create Service Trouble Report
- Diagnose Service Problem
- Report Service Problem
- Survey & Analyze Service Problem
- Track & Manage Service Problem

**Resource Trouble Management**

- Close Resource Trouble Report
- Correct & Resolve Resource Trouble
- Create Resource Trouble Report
- Localize Resource Trouble
- Report Resource Trouble
- Survey & Analyze Resource Trouble
- Track & Manage Resource Trouble

**Service Problem Management**

- Reassign / Reconfigure Failed Service
- Manage Service Restoration
- Implement Service Problem Work Aroun ds
- Generate Service Problem
- Convert Report To Service Problem Format
- Estimate Time For Restoring Service
- Verify Service Configuration
- Perform Specific Service Problem Diagnostics
- Schedule Routine Service Problem Tests
- Stop And Start Audit On Services
- Notify T&M Root Cause Service Problem
- Monitor Customer Problem
- Distribute Customer Problem Notifications
- Distribute Customer Problem Management Reports & Summaries
- Manage Service Alarm Event Notifications
- Filter Service Alarm Event Notifications
- Correlate Service Alarm Event Notifications
- Abate Service Alarm Event Records
- Trigger Defined Service Alarm Action
- Coordinate Service Problem
- Perform First In Service Testing
- Cancel Service Problem
- Escalate/End Service Problem
- Perform Final Service Test

**Resource Problem Management**

- Repair / Replace Failed Resource
- Isolate Unit with Fault
- Manage Standby Resource Units
- Implement Resource Trouble Work Aroun ds
- Generate Resource Trouble
- Convert Report To Resource Trouble Format
- Estimate Time For Restoring Resource
- Verify Resource Configuration
- Perform Specific Resource Trouble Diagnostics
- Perform Specific Resource Trouble Tests
- Stop And Start Audit On Resources
- Schedule Routine Resource Trouble Tests
- Notify T&M Root Cause Resource Trouble
- Monitor Resource Trouble
- Distribute Resource Trouble Notifications
- Distribute Management Reports & Summaries
- Manage Resource Alarm Event Notifications
- Filter Resource Alarm Event Notifications
- Correlate Resource Alarm Event Notifications
- Abate Alarm Event Records
- Trigger Defined Action
- Coordinate Resource Trouble
- Perform First In Testing
- Cancel Resource Trouble
- Escalate/End Resource Trouble
- Perform Final Test
- Engaging External Suppliers