SO Enhancement Points for R2 - a design idea

Byung-Woo Jun, Ericsson

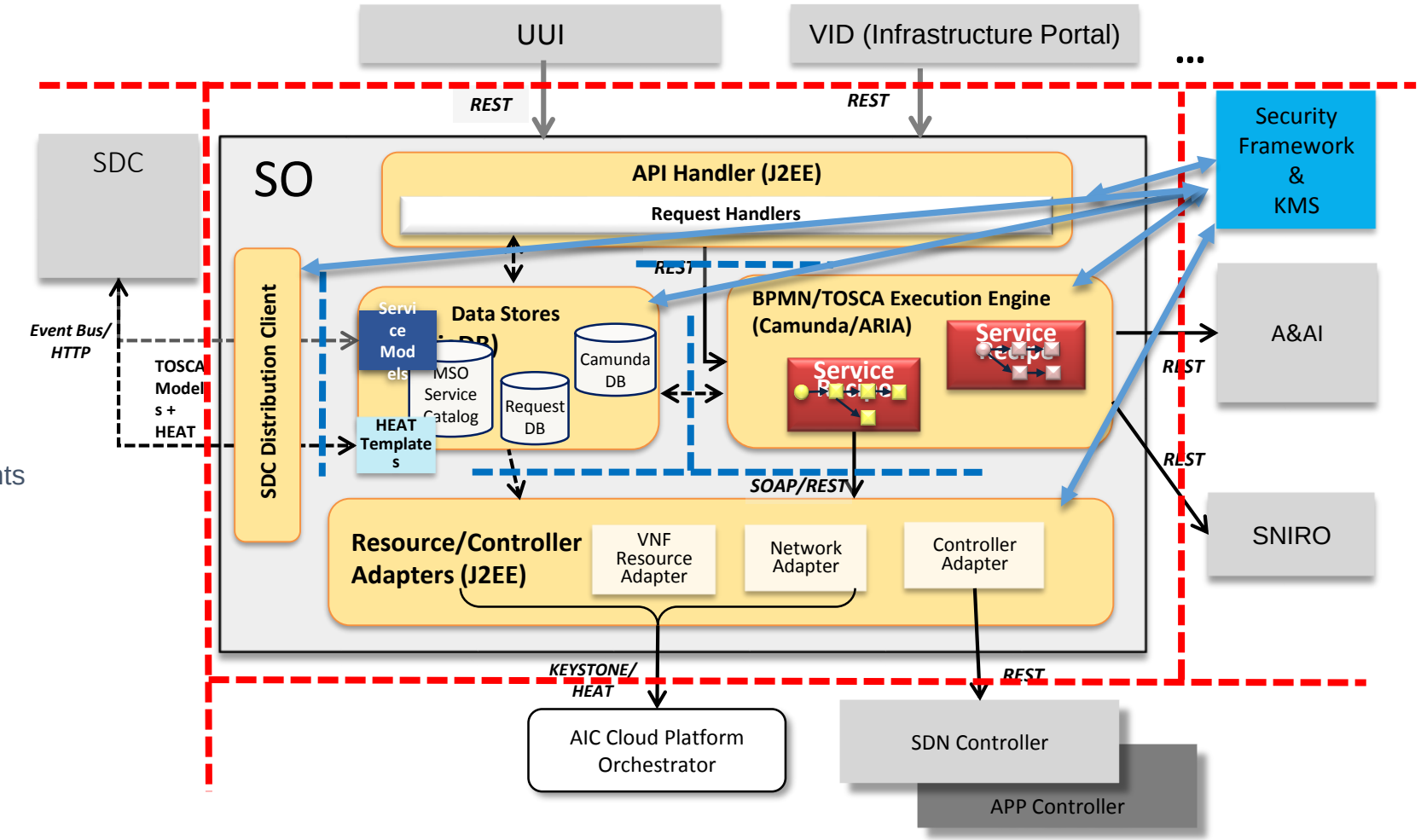
Date , 2018-01-17

SO Carrier-Grade Mission

- To make SO Carrier-Grade, non-functional requirements must be fulfilled. ONAP defined the following non-functional requirements.
  - Scalability (Scale in and out)
  - Stability (managing steady load for required time period)
  - Resiliency (Failover, HA)
  - Security (Secure Communication, AAA, Security Logging/Auditing)
  - Performance (Response Time, Transaction/message rate, Latency, Footprint)
  - Manageability (SSO, Logging/Tracing, Monitoring)
  - Usability (conform to ONAP-level Usability)
- To achieve the above, SO needs platform-level enhancements.
  - SO process monitoring is another important factor from a Carrier-Grade perspective.
    - The monitoring provides manageability (and could be usability).
  - We need to avoid vendor lock-in (e.g., adding commercial products into ONAP).
- This document proposes how SO should approach non-functional requirements and work with other ONAP components.
  - SO will conform to ONAP-level Carrier-Grade design guidelines; which is defined as a dependency.
  - The design concept in this document acts as a starting point. More details will follow.

SO Security (1/2)

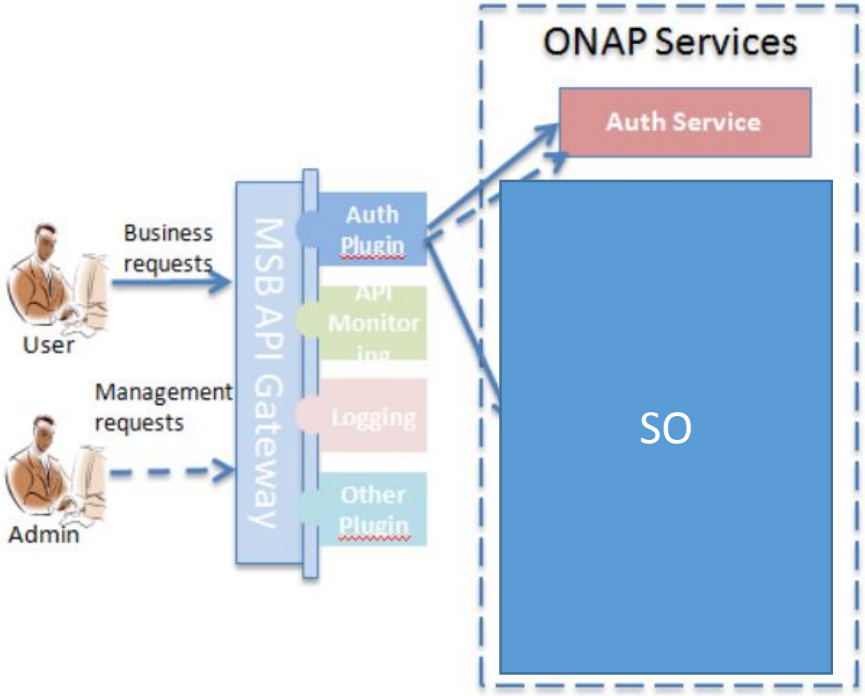
- SO component and its sub-components will be protected.
  - Use mutual authentication for secure communications (encryption)
    - Between SO and external components
    - Between SO sub-components.
    - **Works with the centralized key management system (KMS).**
- User Authentication and Authorization
  - Support Multi-Factor Authentication.
  - Support Role-based Authorization, at least.
  - **By utilizing Security Framework APIs (AAF?)**
  - Support **SSO** across ONAP components and clients applications.
  - User security token will be carried across the SO components.
  - User CRUD management is outside of SO responsibilities.
- SO will validate incoming certificates, credentials/tokens:
  - by utilizing security framework APIs, or
  - Use of interceptor (MSB API Gateway) for token validation
- Support security logging and auditing (see manageability).

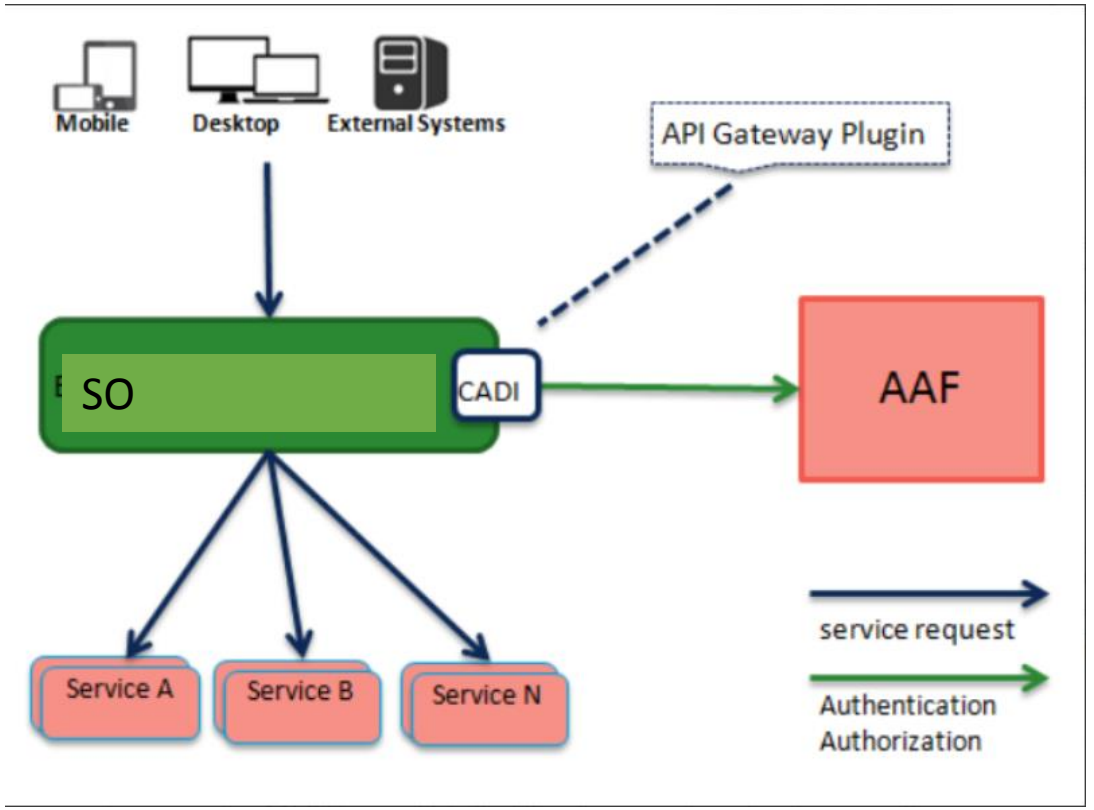


External SO Interface/Access security (red lines)
Internal SO sub-component security (blue lines)

SO Security (2/2)

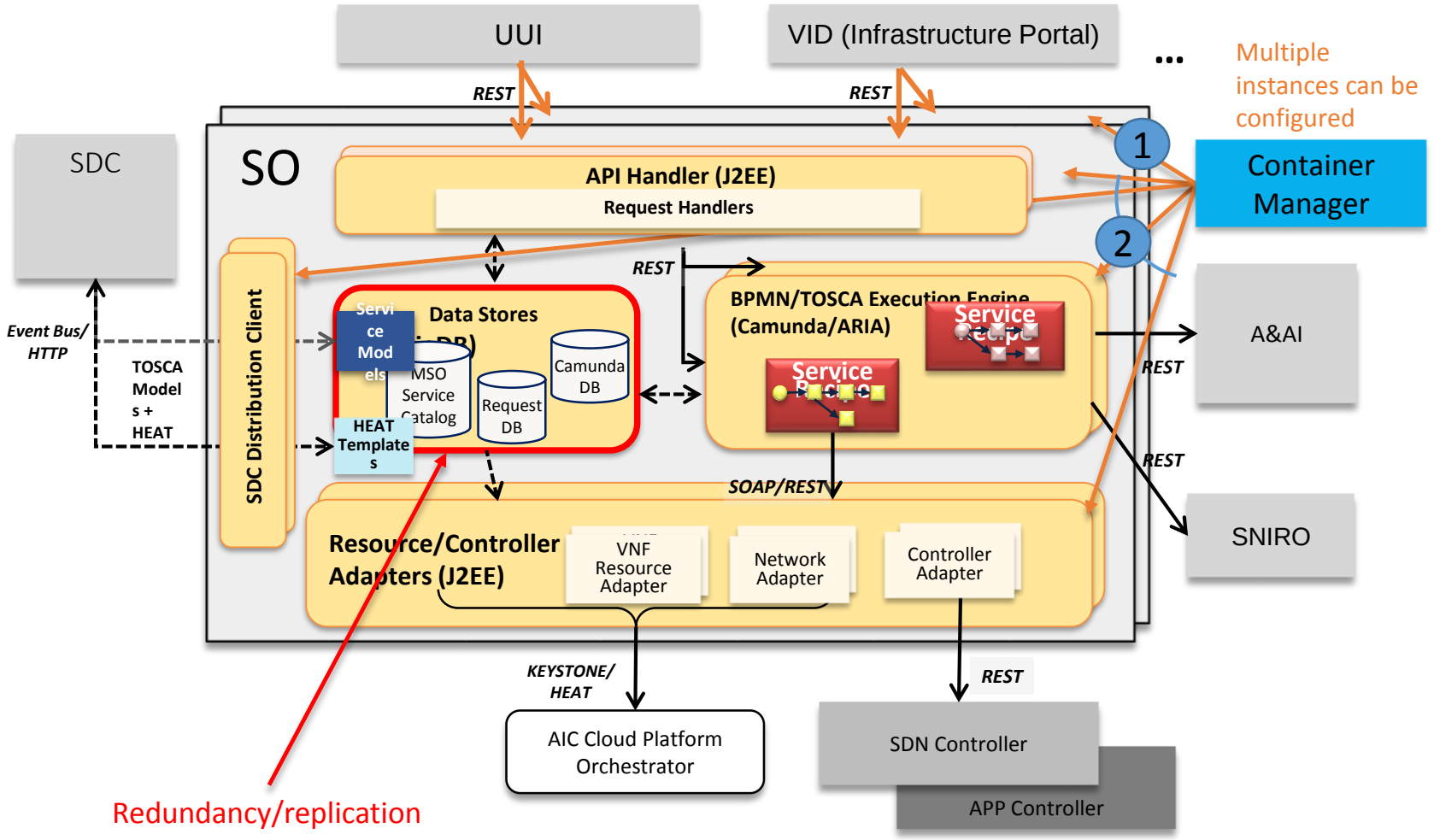
- For authentication, MSB API Gateway and its plugin act as an interceptor, and check authentication with AAF.
- For authorization, MSB API Gateway and its plugin check the user authorization on behalf of SO; however, this is for mostly coarse-grained authorization.
- For fine-grained authorization, SO API Handler can request an authorization decision to the AAF through CADI interfaces.





SO Scalability

- Do we want, 1) SO to be scaled as a unit or 2) its sub-components to be scaled independently?
  - OOM defined MariaDB and SO
- SO and SO sub-components will be packaged and deployed as microservice components to achieve target scalability.
- SO and SO sub-components are able to scale independently based on load traffic.
  - SO can scale independently from other ONAP components.
  - SO sub components can scale independently to achieve target scalability.
  - Container Manager (OOM) will handle the SO scalability.
- The Min and Max will be configured and will be scaled based on predefined policies.
  - For RESTful APIs, Active-Active load balancing will be applied to distribute traffic (e.g., utilizing OOM, CHAP).
  - For the centralized database access, pool and redundancy/replication will be configured (e.g., use of MDBC).

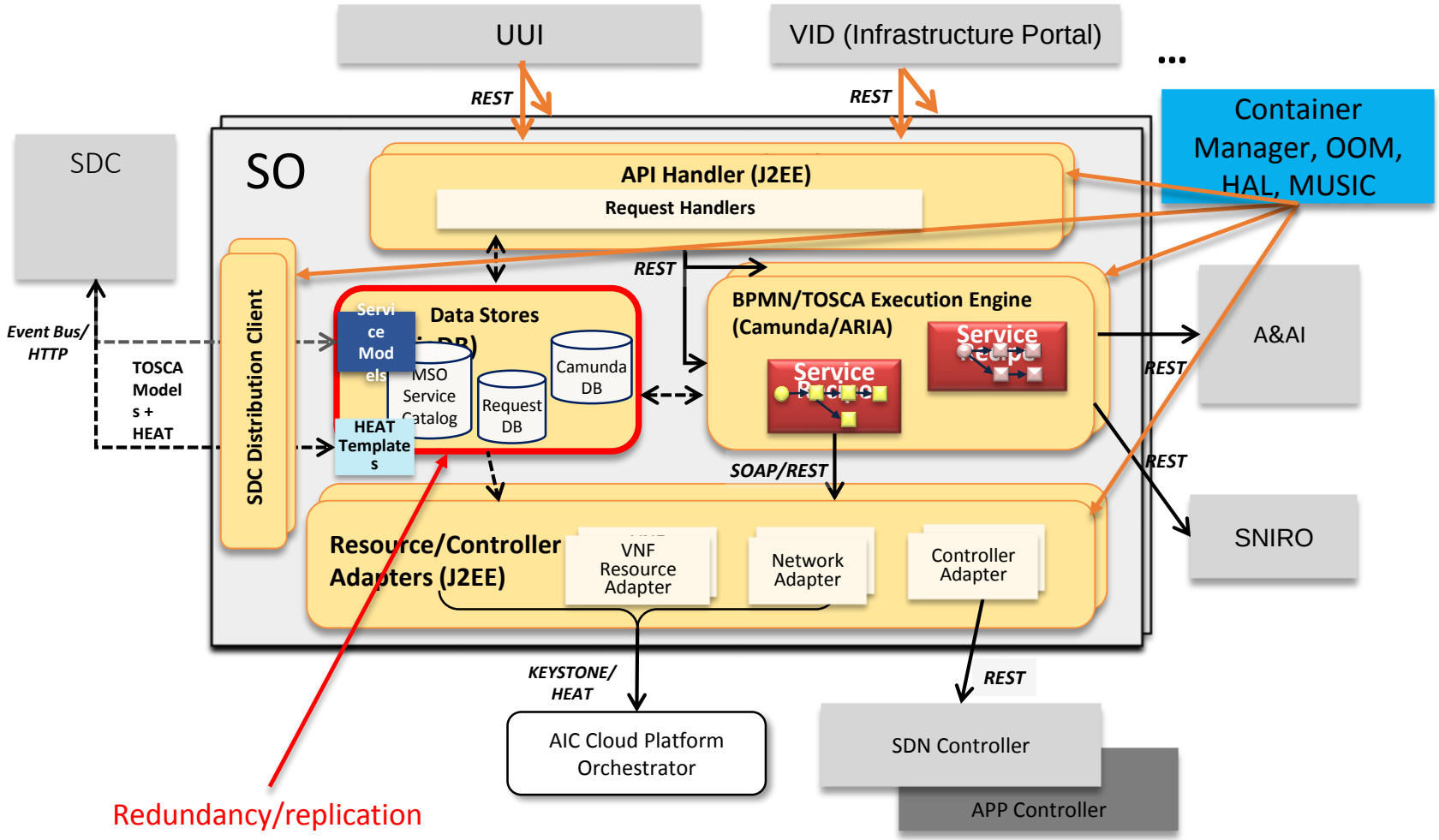


SO Scalability – Requirements…

- SO scalability will be supported by managing multiple SO instances by utilizing OOM.
  - In OOM, the number of SO component instances will be configured to control the number of active SO instances.
  - Each BPMN execution engine will be configured for a shared database, so the engine can be scaled promptly and ready to handle assignments.
- SO instances will be registered to MSB for communication load-balancing.
- SO run-time scalability handling
  - SO will have multiple Camunda Execution engine instances which share the centralized data store.
    - The centralized data store will be replicated, and the replication will be transparent to other SO components.
- The individual execution engine instances do not maintain session state across transactions.
  - The complete state is flushed out to the shared database when a process instance is complete or waiting for events (e.g., asynchronous event, message, human task, etc.).
  - Or, asynchronous continuations can be used during the workflow design when it is necessary to control save points actively (by design) and flush out the process instance states to the database.
  - Once a process instance is passivated, another engine instance will pick up and execute the remaining process instance flows.
- Multiple SDC distribution client instances will be instantiated.
  - A SDC notification will be routed to (or picked up by) one of the SDC notification client instances. Then, the assigned client instance will:
    - query for templates/models from SDC.
    - parse the template/models and store in the Catalog DB.
  - Due to less frequent templates/models changes and SDC notification client activities, a small number of SDC distribution client instances will be configured.
- Multiple API handler instances will be instantiated, and all of the instances are active (active-active).
  - The requests from VID, External API and UUI towards the API handler instances will be distributed/routed via load-balancing. MSB is expected to handle their load-balancing.
  - An assigned API handler will communicate with the orchestration execution engine and Data store in a scalable manner.
    - Communications (invoking BPMN execution) with the orchestration execution engine will be done through MSB, no direct contact with hard-coded endpoints.
    - For storing requests and select recipes, the API Handler will communicate with the Data store (Request DB, Service Catalog), which is replicated.
- Multiple Resource/Controller Adapters will be instantiated for active-active operations.
  - The communications between the BPMN/TOSCA resource recipes and the adapter instances will be load-balanced through MSB.
- External communications with other ONAP components such as DACE, OOF, A&AI, SDNC, etc. will be done through MSB/DMaaP in a scalable manner like the above communication requirements.

SO Resiliency

- SO and SO sub-components are deployed and managed by the container manager such as OOM Kubernetes, HEAT, etc.
- When a microservice SO component in a container is down, the manager will bring up another instance of container.
  - Stateless component approach for SDC Distribution Client, API Handler, BPMN Execution Engine, ARIA and Adapters.
  - Stateful component approach for Data stores with redundancy/replication (e.g., utilizing MUSIC).
- SO components need to be designed to handle failover situations (e.g., HAL, session replication, centralized database and configuration).
  - Camunda Execution Engine supports resiliency.
  - Conform to ONAP HA design guideline (e.g., CHAP).

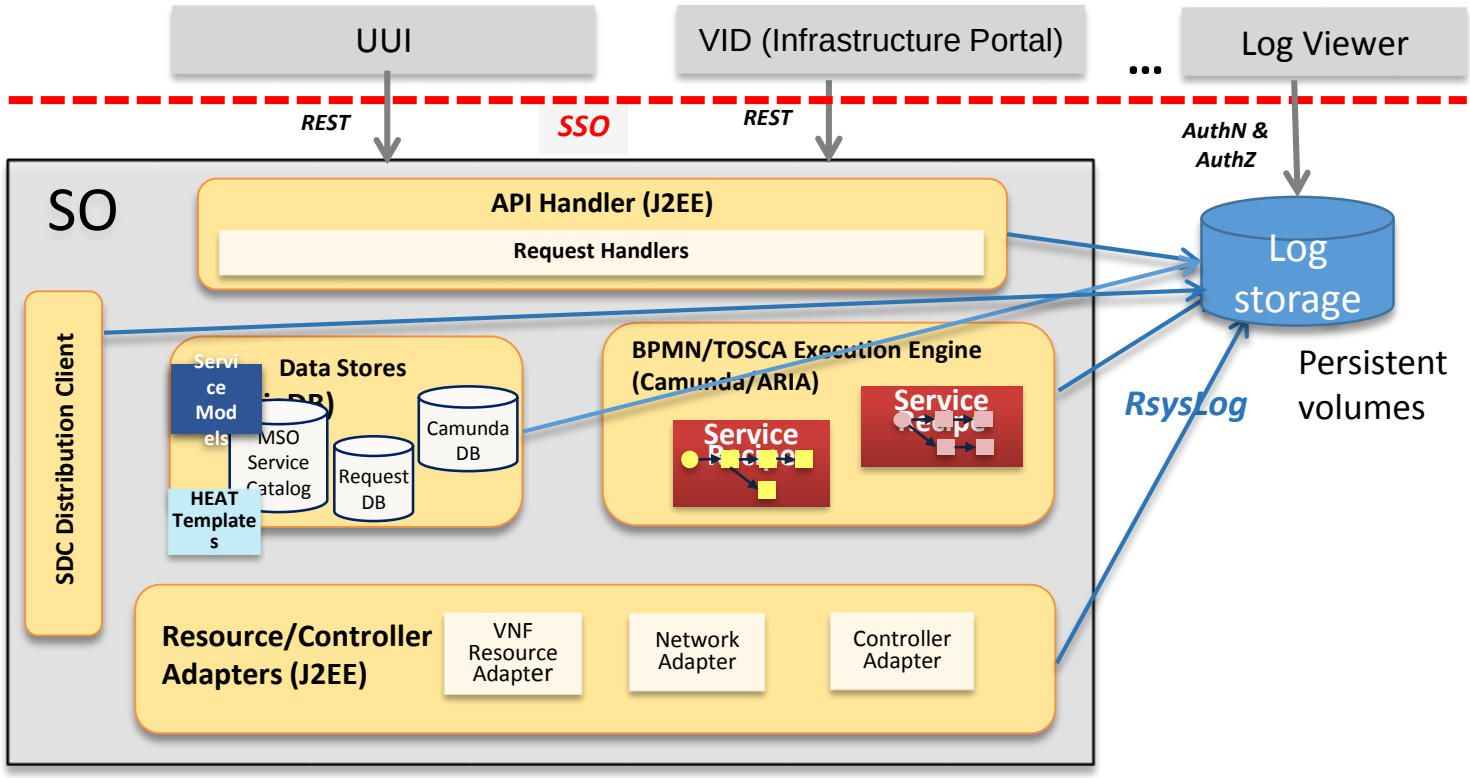


SO Resiliency/Auto Heal - Requirements

- SO resiliency / auto heal will be supported by utilizing OOM.
  - In OOM, the number of SO component instances will be configured.
  - SO component instances in different containers will be configured for fail-over.
  - Each BPMN execution engine will be configured for a shared data store to make the engine instance stateless.
  - OOM will detect failed SO components / containers and create new instances for substitution - cattle-based instantiation.
- SO Resiliency / Auto Heal run-time handling
  - When a microservice SO component in a container is down, the OOM will bring up another instance of component / container.
  - SDC Distribution Client, API Handler, BPMN Execution and Adapters will be stateless for a quick fail-over.
    - Multiple instances of the SDC Distribution Client API Handler, BPMN Execution and Adapters will be instantiated for active-active HA.
  - Fail-over components will pick up and finish the interrupted assignments.
    - In BPMN case, the engine will re-execute the interrupted workflows from the save points.
    - When an entire workflow was abandoned, the API Handler will retry the requests.
  - The centralized data store components will be replicated to avoid single-point-failure
    - Master-Slave data replication
    - by utilizing MUSIC
    - Service Catalog, Camunda DB and Request DB data will be replicated.
  - The individual execution engine instances do not maintain session state across transactions.
  - The complete state will be flushed out to the shared database.
  - Asynchronous continuations will be used when it is necessary to control save points actively and flush out the process instance state. So, another process instance can pick up the remaining process instance flows.

SO Manageability

- SSO support (It is part of Security and will facilitate manageability).
  - Accessing SO does not require separate login.
  - VID/UUI authenticated user can send a request toward SO without logging in to SO separately.
  - SO components participate in ONAP SSO by validating incoming security tokens.
  - Centralized security framework will be used for validation.
- Logging and tracing
  - All the SO components generate logging/tracing data, for example:
    - Input and output, Time stamp, Requester/Client
  - The data will be collected by remote and centralized logging storage thru Rsyslog and LogStash, for example. The reason is a container is ephemeral.
  - SO Monitoring capability will facilitate the SO Logging and tracing.
  - Log Viewer widget will be developed at the ONAP level.
    - Provide filters (define search criteria)
    - Its access will be under control based on authentication and authorization with SSO.



SO Monitoring (a design idea)

- Why not Camunda Cockpit as is: current Camunda Cockpit was designed from a BPMN process management perspective (note: need to study for TOSCA cases)
  - It does not meet service-level orchestration monitoring.
  - It is designed for BPMN definition/execution monitoring; Requires process knowledge for monitoring
- We need higher-level monitoring abstraction for both BPMN and TOSCA.
  - Associate Service Instance id (or other keys) to the top-level process instance id
    - Could use a process variable holding the Service Instance id (or other keys), or
    - Could use a database holding the association
  - Allow VID or UUI or external apps monitor process workflow progress (graphically and text-based) based on search keys.
- We need a platform level runtime and history process activity report capabilities out of the box.
  - Even without Camunda Enterprise Edition
  - Query to Camunda/ARIA Database to extract activities.

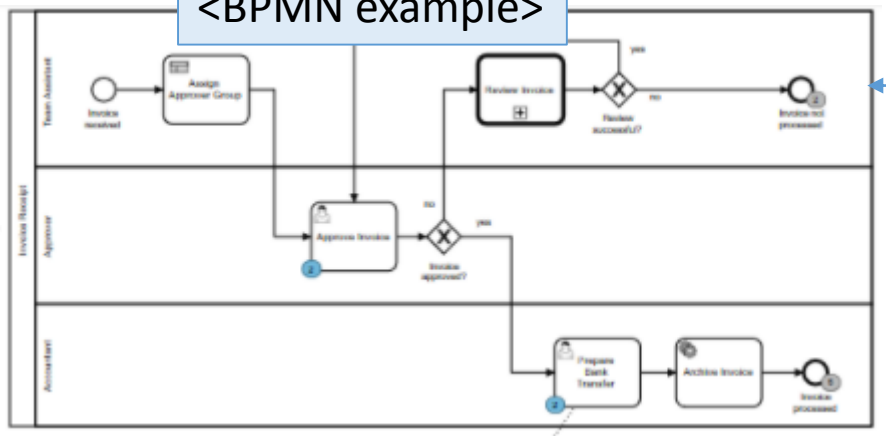
VID (Infrastructure Portal)

UUI

External Applications

Invoke process report
Based on search criteria (service id, request id) without knowing workflow definitions

<BPMN example>



A new widget that will be launched from VID, UUI, Ext Apps based on key mapping

- This widget utilizes Camunda Cockpit APIs for queries.

SO Monitoring (Rationale) ++

- Search is an Camunda enterprise feature, but we need to provide searching capability for non-enterprise edition.
  - Finding right process instance(s) for a VNF service request is tedious and hassle.
  - To facilitate monitoring, we need more than what Camunda Community/Enterprise edition supports
  - Provides the process monitoring (instance-search) hyperlink to the SO clients for launching process monitoring.
    - Automates tedious manual steps for finding target process instance(s)
    - Access customized SO Service List/ Camunda cockpit widgets from VID, UUI and external Apps.
- If a service provider uses Camunda Enterprise edition, they can still utilize this SO monitoring on top of Camunda Enterprise edition features.
- Many of Camunda Enterprise features such as CRUDV and version control of process definitions would be part of SDC.
  - ONAP separated workflow design and runtime .
  - Several Enterprise features are not part of SO monitoring features and are not applicable.
- What are the current TOSCA Orchestrator monitoring capabilities?
  - So monitoring should cover both imperative and declarative orchestration
  - Can we have a kind of uniform way of monitoring?
- Alternatively, should we SO make monitoring part of the ONAP-level monitoring, and not to specific to Camunda?
  - In that case, SO just needs to populate the logging data, etc.

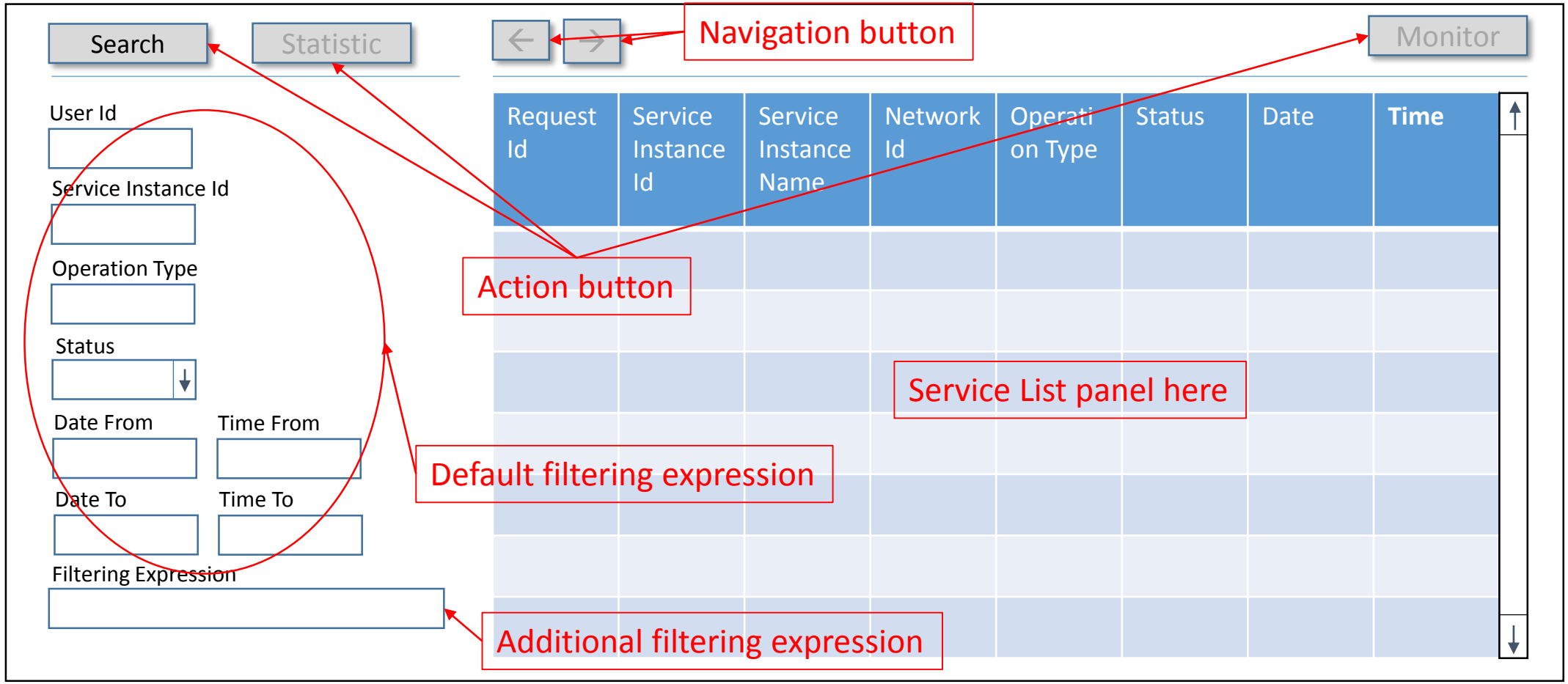
SO Monitoring (Requirements) ++

- Dashboard views of Service lists
  - Filtering Capabilities based on search criteria
  - Configurable search criteria
- Dashboard views of statistics (donuts, pie charts, etc.) for filtered service instances
- Service Instance views
  - with sub-service instance drill-down and drill-up capabilities
    - A service instance could be realized by multiple process instances
  - With process / Task detail
  - Topology (workflow) views during/after orchestration
- Input/output data views for process/task/service task (messages, parameters)
  - Display on service / task detail panel
  - Provide Message Log views (could be on a pop-up widget)
- Color coding/visual indication of statistic and service type and status
- Troubleshooting capability by manipulating the workflow during orchestration for troubleshooting and retry from the current location (stretch goal)
- TOSCA orchestration monitoring (stretch goal)

SO Monitoring (Service List widget) ++

- Provides monitoring capabilities for processed services based on search criteria
  - Configurable Search Criteria filtering: Service ID, Operation Type, Status, User Id, Date/Time range
  - Actual filtering criteria fields could be changed based on configuration

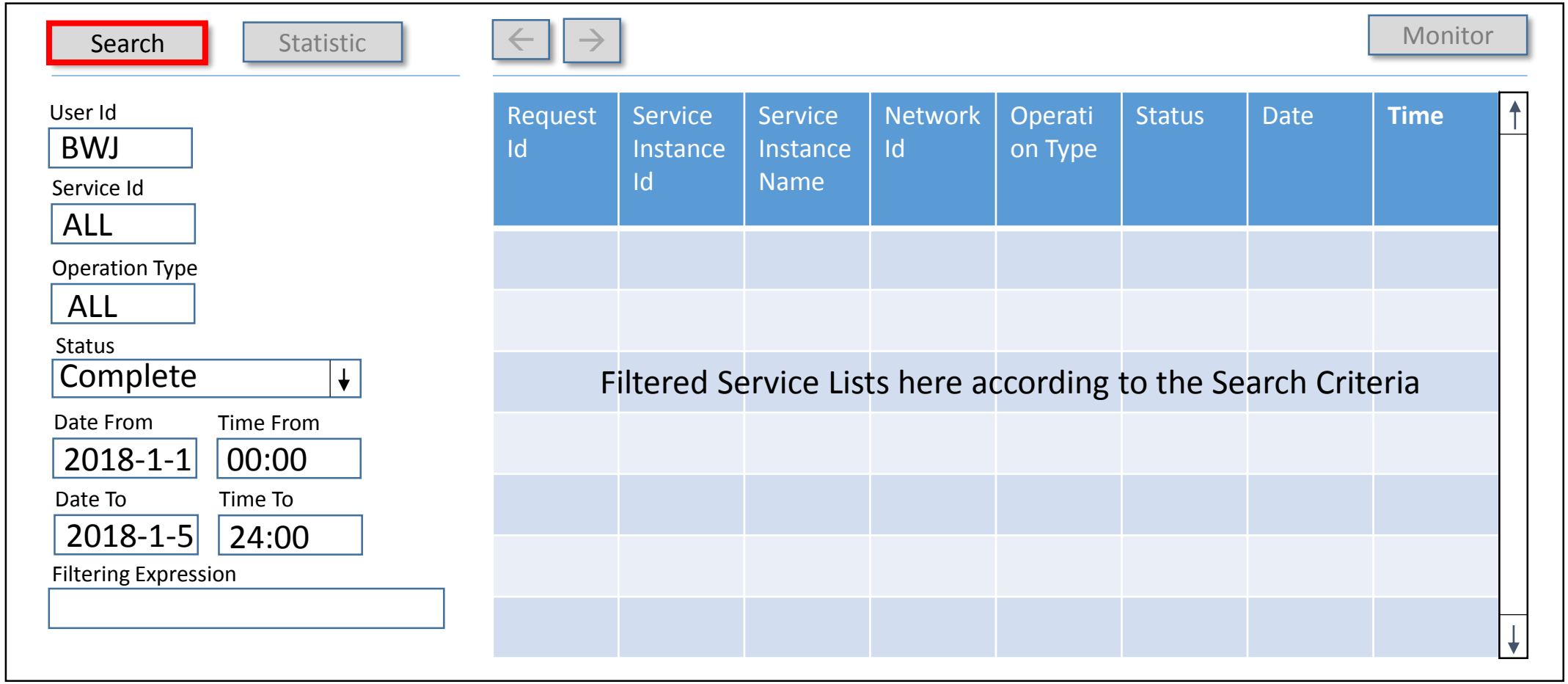
** mockup **



SO Monitoring (Service List widget) ++

- Utilize SO existing Request DB APIs to get the Service list.
  - Provides filtering, Service ID, Operation Type, Status, User Id, Date/Time range (actual criteria could be changed)
  - Click the Search button, and the filtered service list will be displayed

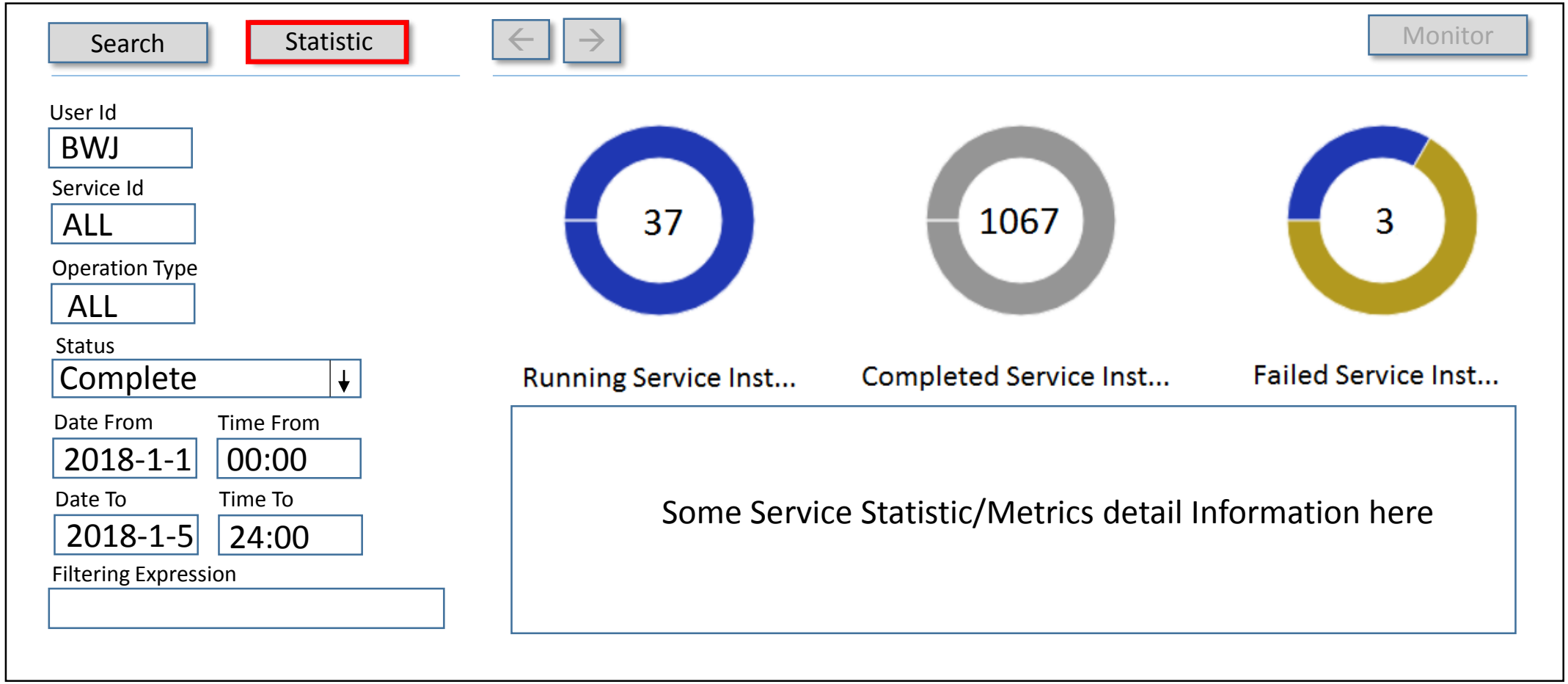
** mockup **



SO Monitoring (Service Statistic widget) ++

- Service Statistic
  - Service Statistic Dashboard will be shown from the Service perspective.
  - Collect service instance statistic per filtered service list.

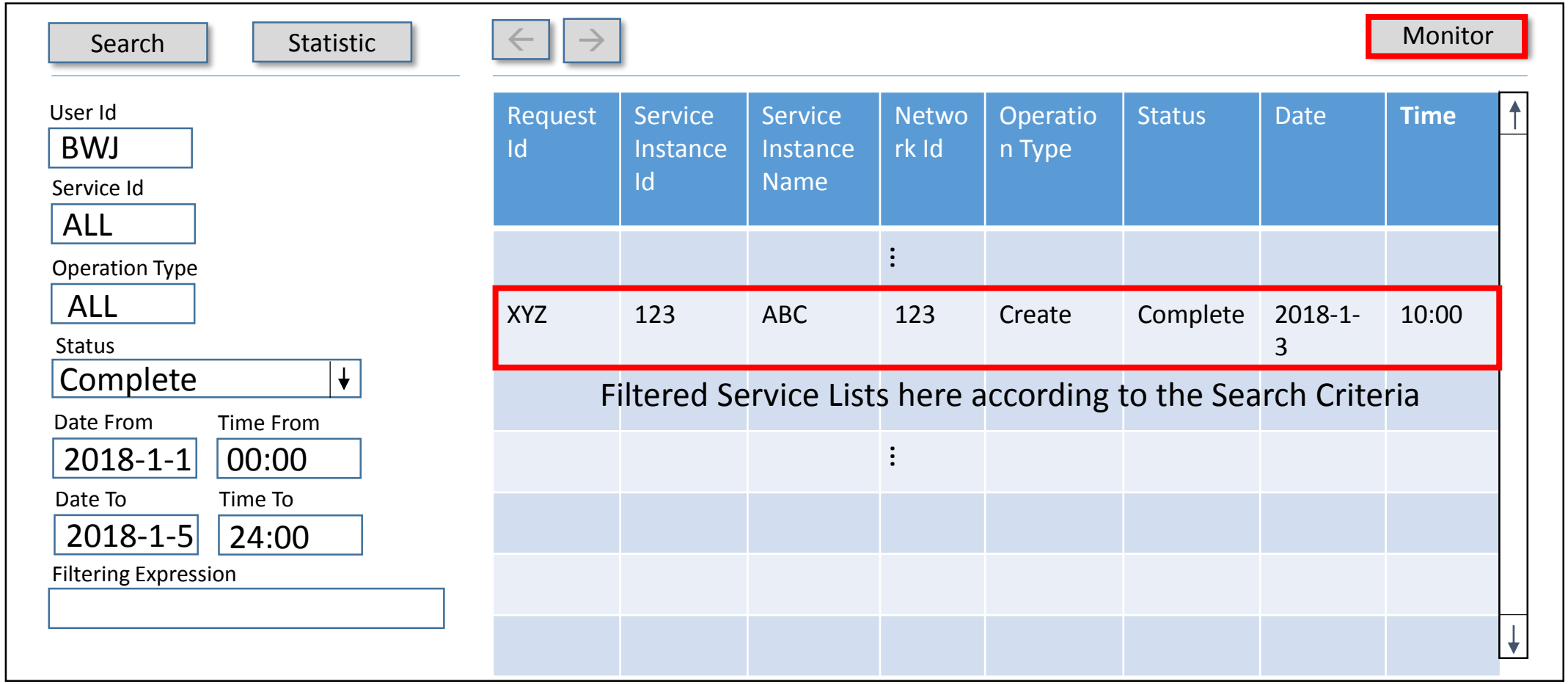
** mockup **



SO Monitoring (Service List widget) ++

- Get the Process Instance detail
  - Select a row and click the Monitor button. Then, it will go to the Service instance (=process instance in Camunda) monitoring widget for Service instance {XYZ}.

** mockup **



SO Monitoring (Service Instance Rendering & Detail View) ++

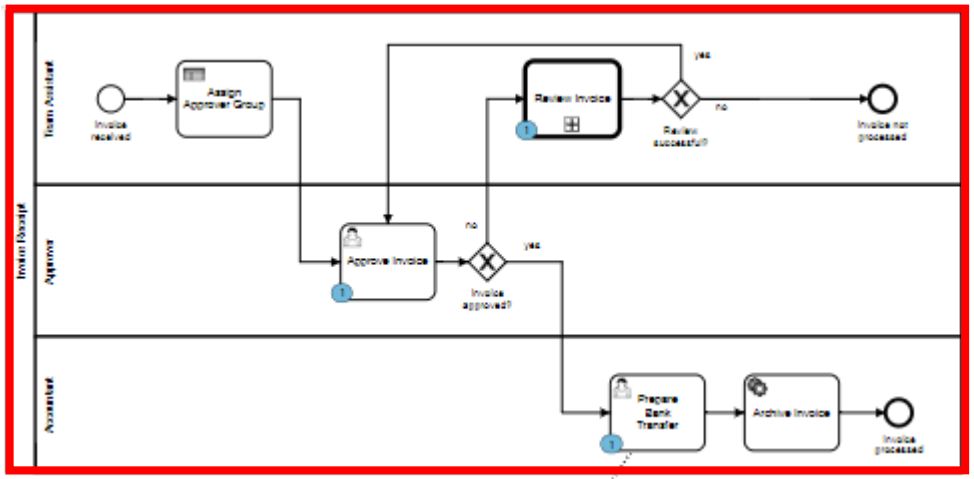
- Get a process definition XML through the Service Id and Process Instance association, get /process-definition/{id}/xml
- Get the state of a process instance from the activity-instances, get/process-instance/{id}/activity-instances
- Render the BPMN XML with bpmn.io and places markers on top of it, and provide Service instance detail views

** mockup **

User Id	Service Id	Operation Type	Status	Date From	Time From	Date To	Time To
BWJ	XYZ	CREATE	Complete	2018-1-1	00:00	2018-1-5	24:00

Filtering Expression

Drill-Down | Drill-Up | Close



- Display Service instance activity details with the current status

Service Instance XYZ Activity Details			
Display Service instance XYZ detail			

SO Monitoring (Task Drill-down/-Up & Detail View) ++

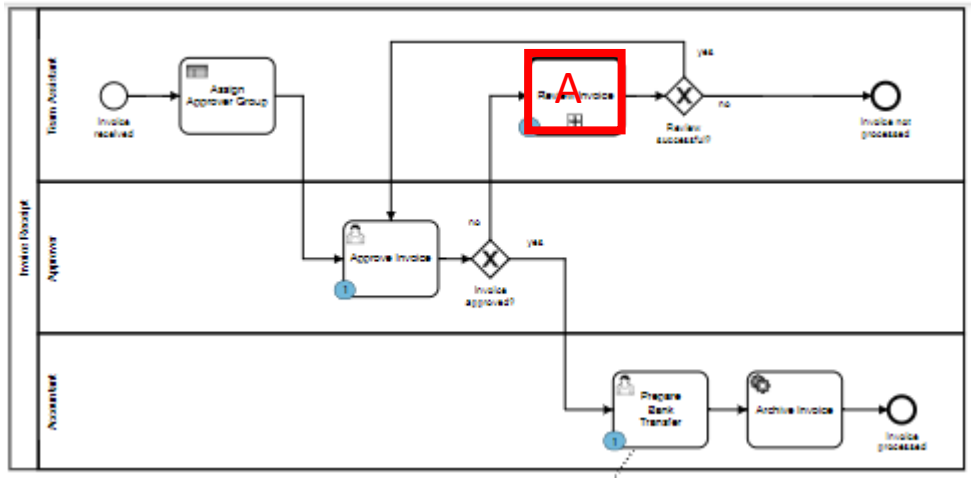
- Support task (Call Activity) drill-down/drill-up capabilities
- Provide Task detail views

** mockup **

User Id	Service Id	Operation Type	Status	Date From	Time From	Date To	Time To
BWJ	XYZ	CREATE	Complete	2018-1-1	00:00	2018-1-5	24:00

Filtering Expression

Drill-Down | Drill-Up | Close



- When clicking on a task, display the detail
- When clicking the 'Drill-Down', go to its sub-service instance flow

Task A Activity Details

Display Task A detail (input and output data, variables)

SO Monitoring (Sub-Service Instance Rendering) ++

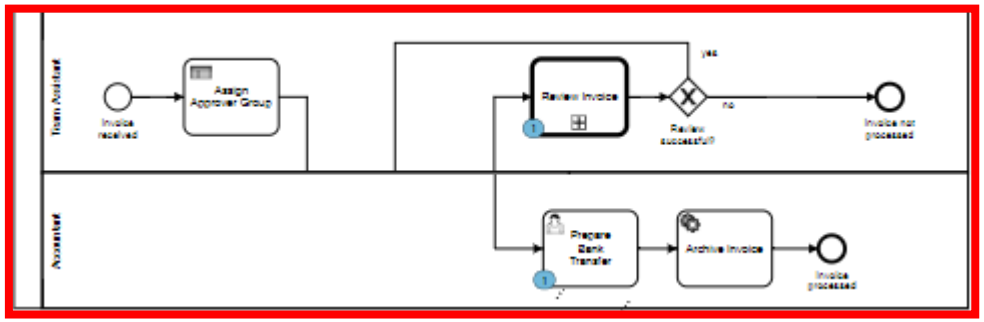
- Get a sub process definition XML through the Service Id and Process Instance association, get /process-definition/{sub-id}/xml
- Get the state of a process instance from the activity-instances, get/process-instance/{sub-id}/activity-instances
- Render the BPMN XML with bpmn.io and places markers on top of it, and provide service instance detail views

** mockup **

User Id	Service Id	Operation Type	Status	Date From	Time From	Date To	Time To
BWJ	XYZ	CREATE	Completed	2018-1-1	00:00	2018-1-5	24:00

Filtering Expression

Drill-Down | Drill-Up | Close



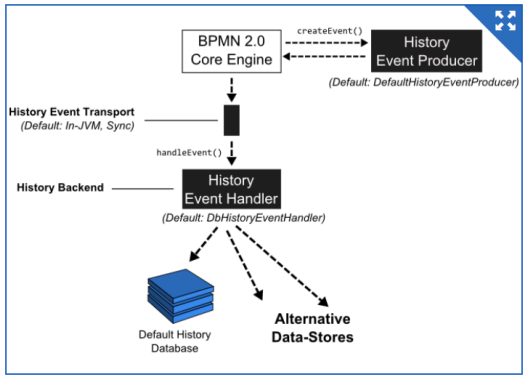
- Display Service instance activity details with the current status
- When a task is selected, and click the ‘Drill-Down’ button, it moves to its sub-process.
- Use Drill-Up to go back to its parent process

Service Instance A Activity Details			
Display Service instance A detail			

SO Monitoring (High-level Design) ++

- SO Service list widget
  - Queries a service list from the SO Request DB (display key fields in the list)
  - The widget can be triggered from other UIs (VID, UUI, external Apps)
  - From here, invokes process monitoring; bypassing tedious search starting from the process definitions.
  - Possibly, link to TOSCA monitoring as part of e2e monitoring
- Statistic widget
  - It is a Service Statistic Dashboard that will show statistic from the Service perspective.
  - Collect service instance statistic per filtered service list.
- Service Instance Rendering and Detail Panel
  - Get a process definition XML through the Service Id and Process Instance association, get /process-definition/{id}/xml
  - Get the state of a process instance from the activity-instances, get/process-instance/{id}/activity-instances
  - Render the BPMN XML with bpmn.io and places markers on top of it, and provide Service instance detail views.
- REST APIs for providing data to the above UIs.
  - Provides REST services, by utilizing Camunda REST APIs, such as BPMN XML string, process activity data, process variables, statistic,
  - Consolidate data responses from multiple Camunda calls and feed them to UIs.
  - Use of HistorService APIs; example, processEngine.getHistoryService().createHistoricProcessVariableQuery().xxx
  - Set a History level to ACTIVITY as a minimum; AUDIT (default) level for process variable tracing
  - Provides workflow tracing (between parent-child workflows, interaction with other services; service task in and out); example,
    - processEngine.*getRuntimeService*().createExecutionQuery()

      .processVariableValueEquals("serviceInstanceId", serviceInstanceId).singleResult();
  - Custom Query
    - Custom Query against History ACT_HI_DETAIL database table, as needed
- Custom History Event producer Producer
  - Populate additional data in history with extensibility.



SO Monitoring (Implementation Brainstorming) ++

- Getting History Data
  - To build a process instance list, Use of Camunda History REST APIs, https://docs.camunda.org/manual/7.8/reference/rest/history/process-instance/get-process-instance-query/
  - To get process instance detail, https://docs.camunda.org/manual/7.8/reference/rest/history/process-instance/get-process-instance/
  - To get process instance detail with better filtering, https://docs.camunda.org/manual/7.8/reference/rest/history/process-instance/post-process-instance-query/
- Rendering
  - To get process definition diagram, https://docs.camunda.org/manual/7.8/reference/rest/process-definition/get-diagram/
  - Get a process definition XML through the Service Id and Process Instance association, get /process-definition/{id}/xml
  - Get the state of a process instance from the activity-instances, get/process-instance/{id}/activity-instances
  - Render the BPMN XML with bpmn.io and places markers on top of it.
  - Attach overlay events on the call activities for drilling down and displaying details.

```
var BpmnViewer = require('bpmn-js');

var xml = getBpmnXml(); // get the process xml via REST
var viewer = new BpmnViewer({ container: 'body' });

viewer.importXML(xml, function(err) {

  if (err) {
    console.log('error rendering', err);
  } else {
    console.log('rendered');
  }
});
```

viewer.get('overlays').add(…);

```
var overlayHtml = $('<div>Mixed up the labels?</div>');

overlayHtml.click(function(e) {
  alert('someone clicked me');
});

// attach the overlayHtml to a node
overlays.add('SCAN_OK', {
  position: {
    bottom: 0,
    right: 0
  },
  html: overlayHtml
});
```

SO Monitoring (Next Step) ++

- A little PoC for rendering Service instance graphically and collecting detail
- Estimates
- …

SO Monitoring (Estimate) ++

***** Rough Estimate – to be refined *****

Component	Development Estimate *	Comments
SO Service List widget	80 hours / UI developer	Displays service list
Statistic Dashboard	40 hours / UI developer	Displays Statistic
Service Instance rendering and detail panel	120 hours / UI developer	Service Instance rendering and detail panel
REST APIs for supplying data	80 hours / Java developer	Façade 1) for collecting and consolidating various Camunda process and history data to simplify GUI data collection interactions, and 2) for collecting request DB data
Custom History Producer	40 hours / Java developer	Produce history with additional data population

* Only includes development time with unit testing.

SO Performance

- SO will conform to ONAP performance requirements. The following will be considered during the design and testing:
  - Response Time
  - Transaction/message rate
  - Latency
  - Footprint
- Camunda 7.8 enhanced their database performance by grouping database queries. We may want to consider the upgrade.
- ONAP Benchmark can considered for performance (and other) tests to validate SO performance.

Open Issues

- ONAP Carrier-Grade requirements are not finalized. That is the dependency.
  - How will OOM, CHAP, MUSIC projects impact ONAP components?
  - Need to decide SO scalability strategy: scale at the SO component level or at the SO sub-component level.
  - Need to define the Container Manager functionality and realization.
  - Need to define the Security Framework functionality and realization.
- Dependences
  - OOM / Kubernetes
  - CHAP
  - AAF
  - KMS

Camunda Enterprise Edition Feature

- Camunda Enterprise Edition Feature list
  - https://camunda.com/products/cockpit/#/features