SO Containerization and Evolution Effort

Steve Smokowski & AT&T SO Team

3/25 , 2017

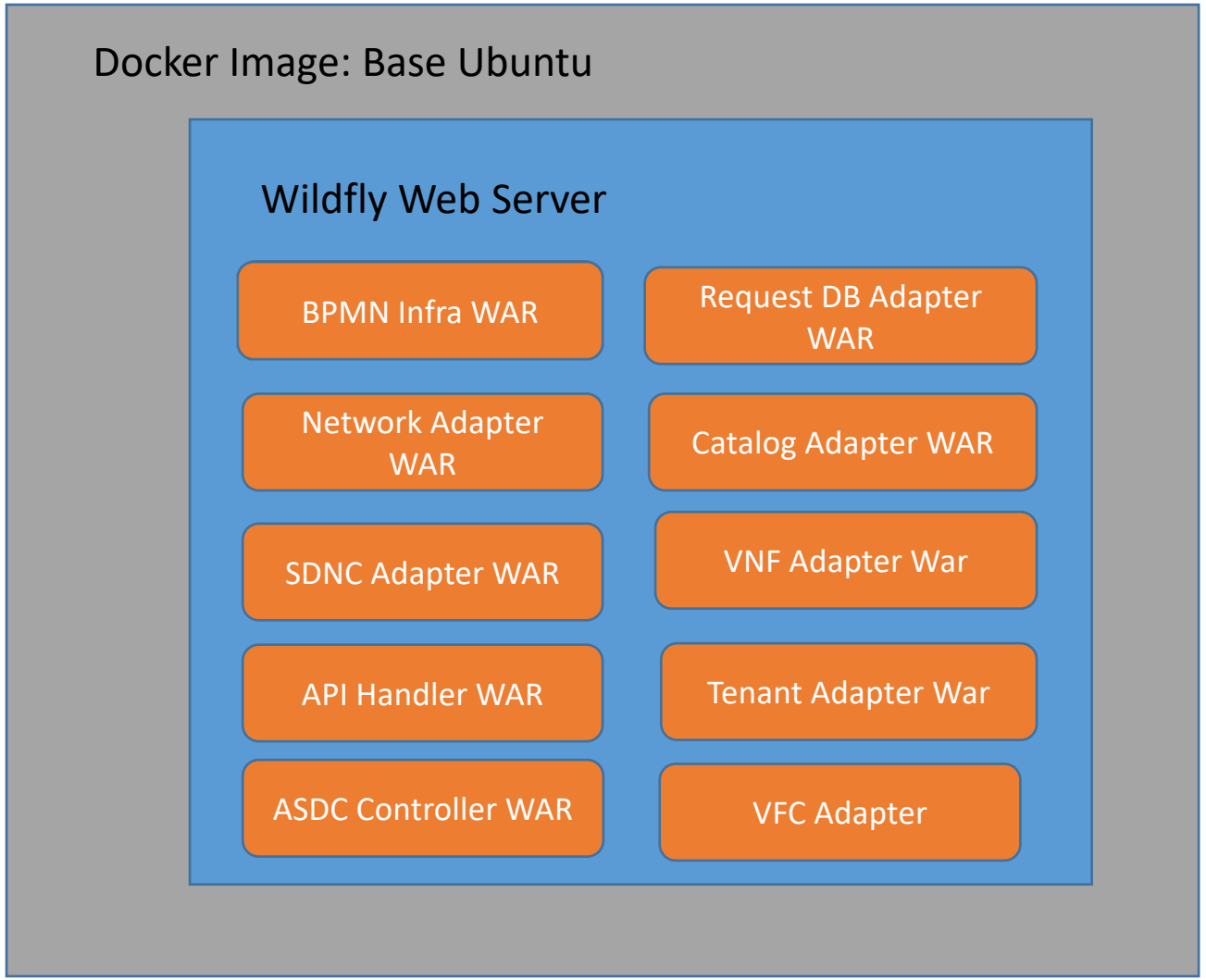
Current Deployment View

-Single webserver

-Individual WAR deployments

-Scale of each WAR effects another

-Entire Java EE Stack



New Deployment View

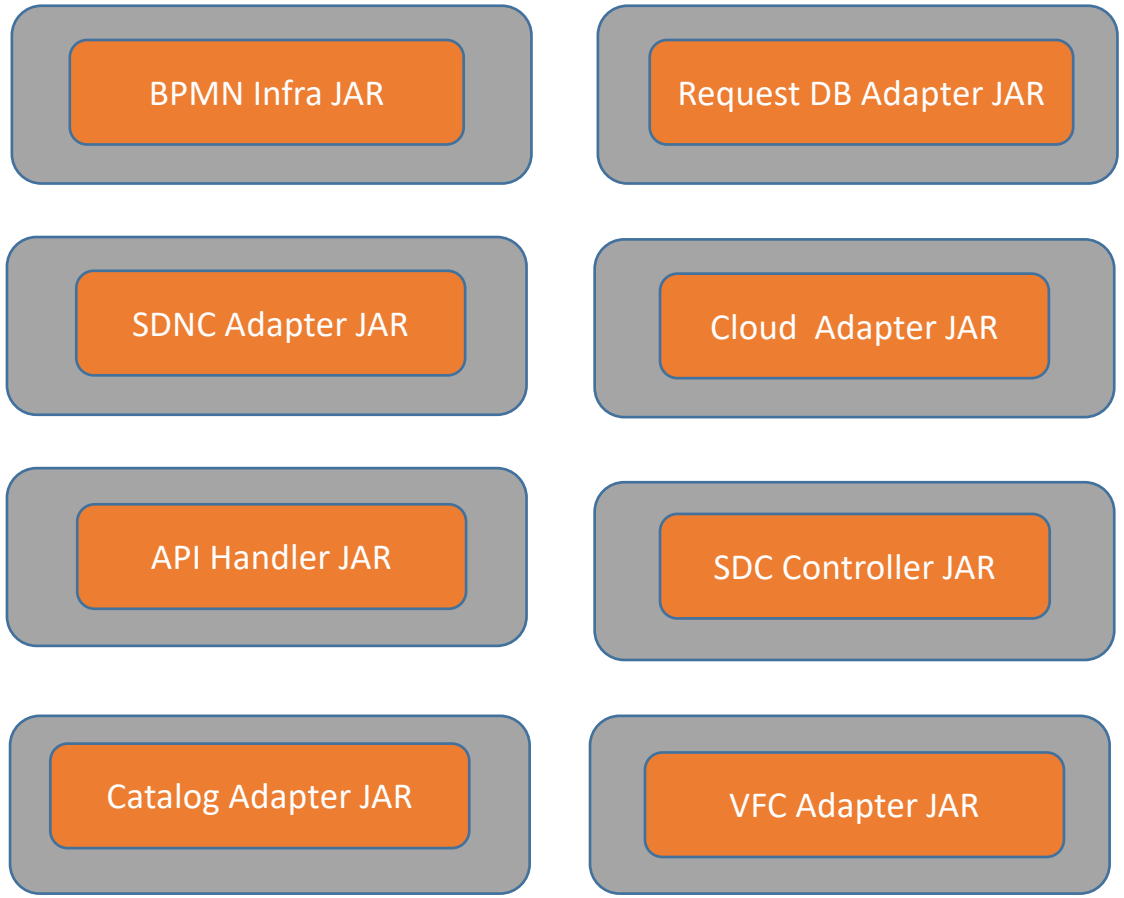
-Base Image uses Alpine Linux

-Replace Wildfly with Tomcat

-Individual Docker images

-Individual deployment/scaling capability

-Utilizing SpringBoot Stack



Current Database Access Implementation

- Hibernate Configuration in XML
- Current Queries use HQL or Hibernate Criteria queries
- Entities are not properly configured to join in Hibernate
- Current service_to_resource_customization mapping table not using foreign keys

```
<hibernate-mapping package="org.openecomp.mso.db.catalog.beans">
    <class name="Service" table="SERVICE">
        <meta attribute="class-description">
            This class describes a Service that may be orchestrated
        </meta>

        <id name="modelUUID"                    column="MODEL_UUID"                type
        <property name="modelName"              column="MODEL_NAME"                type
        <property name="version"                column="MODEL_VERSION"             type
        <property name="description" column="DESCRIPTION" type="string" length="1200"/>
        <property name="toscaCsarArtifactUUID"  column="TOSCA_CSAR_ARTIFACT_UUID"  type

        <property name="created" type="timestamp" generated="insert" update="false" inse
          <column name="CREATION_TIMESTAMP" default="CURRENT_TIMESTAMP"/>
        </property>
        <property name="modelInvariantUUID" type="string">
           <column name="MODEL_INVARIANT_UUID" default="'MANUAL_RECORD'" not-null="true"
        </property>
        <property name="serviceType"                column="SERVICE_TYPE"
        <property name="serviceRole"                column="SERVICE_ROLE"
        <property name="environmentContext"         column="ENVIRONMENT_CONTEXT"
        <property name="workloadContext"            column="WORKLOAD_CONTEXT"


        <map name="recipes" inverse="true" cascade="all">
            <key column="SERVICE_MODEL_UUID"/>
            <map-key column="action" type="string"/>
            <one-to-many class="ServiceRecipe"/>
        </map>

        <set name="serviceResourceCustomizations" inverse="true" cascade="all">
            <key column="SERVICE_MODEL_UUID" not-null="true" />
            <one-to-many class="ServiceToResourceCustomization" />
        </set>
    </class>
</hibernate-mapping>
```

Updated Database Access Implementation

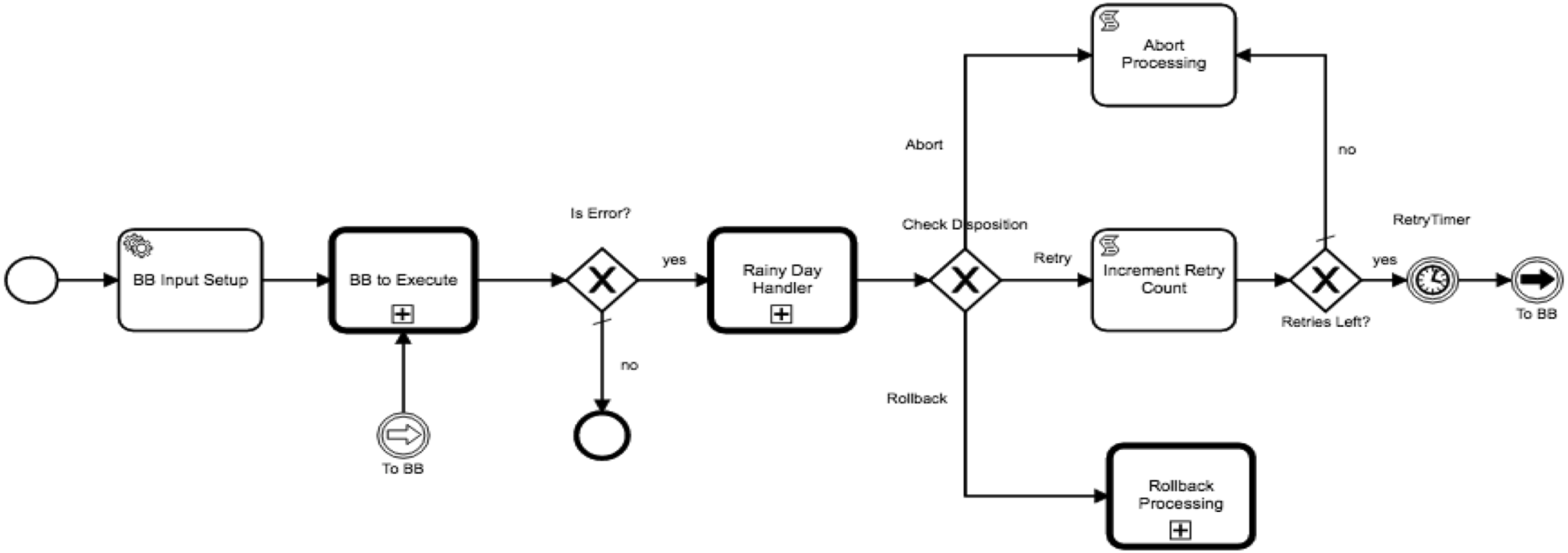
- JPA Annotation based Implementation
- Spring Data Repositories, HQL, JPA criteria queries
- All Entities joined using lazy fetch
- Database migration scripts provided
- All Equals/Hash code methods implemented consistently

```
@Column(name = "ENVIRONMENT_CONTEXT")
private String environmentContext;

@Column(name = "WORKLOAD_CONTEXT")
private String workloadContext;

@OneToMany(cascade=CascadeType.ALL)
@JoinTable(name="network_resource_customization_to_service",
          joinColumns = @JoinColumn( name="service_model_uuid"),
          inverseJoinColumns = @JoinColumn( name="resource_model_customization_uuid")
    )
private List<NetworkResourceCustomization> networkCustomizations;



@OneToMany(cascade=CascadeType.ALL)
@JoinTable(name="vnf_resource_customization_to_service",
          joinColumns = @JoinColumn( name="service_model_uuid"),
          inverseJoinColumns = @JoinColumn( name="resource_model_customization_uuid")
    )
private List<VnfResourceCustomization> vnfCustomizations;
```

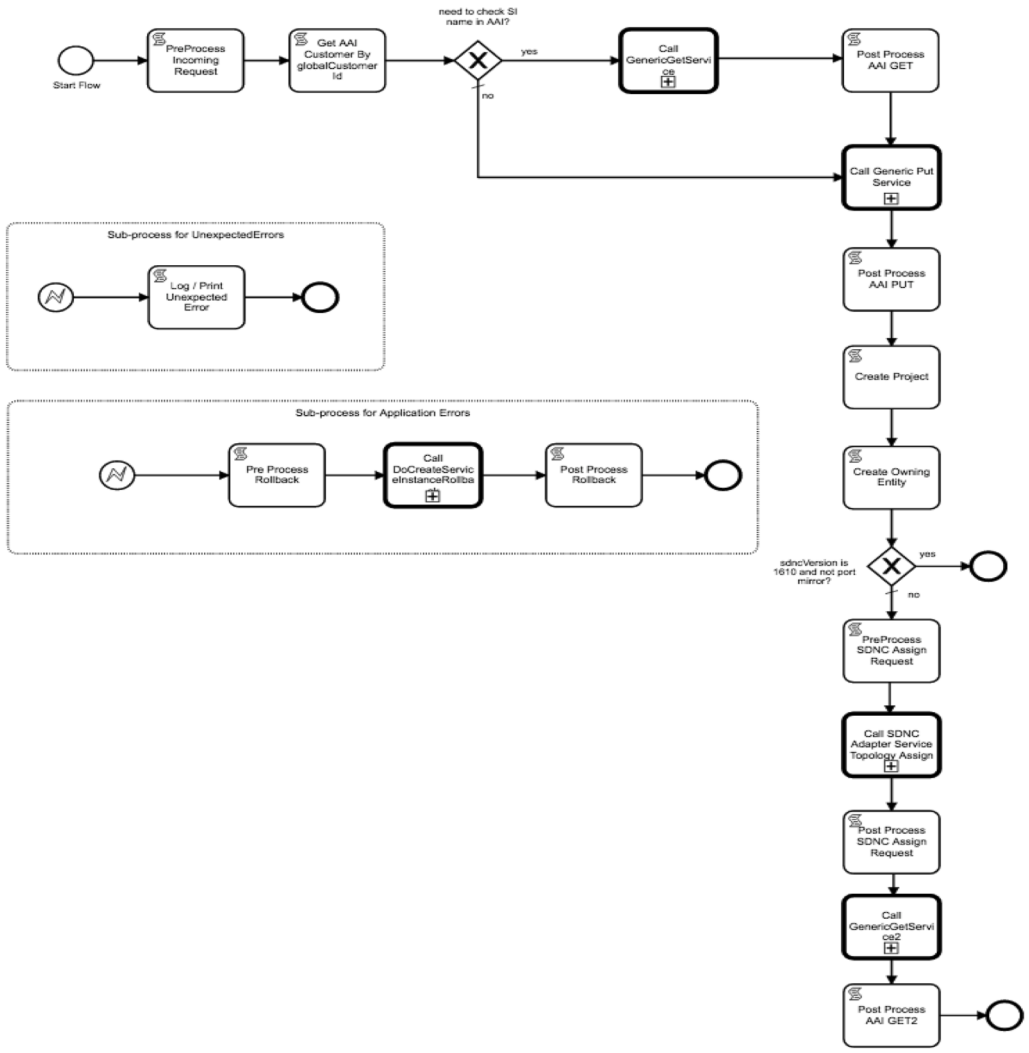
BPMN Building Block Updates

- API consolidation inside SDNC, driving updated building blocks
  - SDNC Generic Resource API will support all L1-L7 VNFs without requiring MSO to have VNF specific logic
  - Scope: service-instance, network, VNF, vf-module, and volume-group
- Orchestration-status standardization:
  - Common/standard state transitions for all service & resources
  - MSO is the owner of the orchestration-status, creating ‘shell’ object in A&AI and updating status
  - Standard method of executing building blocks
- Abstract ExecuteBB BPMN responsible for executing and handling rainy day fall outs for all building blocks
  - Increase ease of re-use
  - Allow for generic fall out handling
  - Allow for easier orchestration of decomposed resources
  - Allow for customizable Macro flows

Execute BB Flow

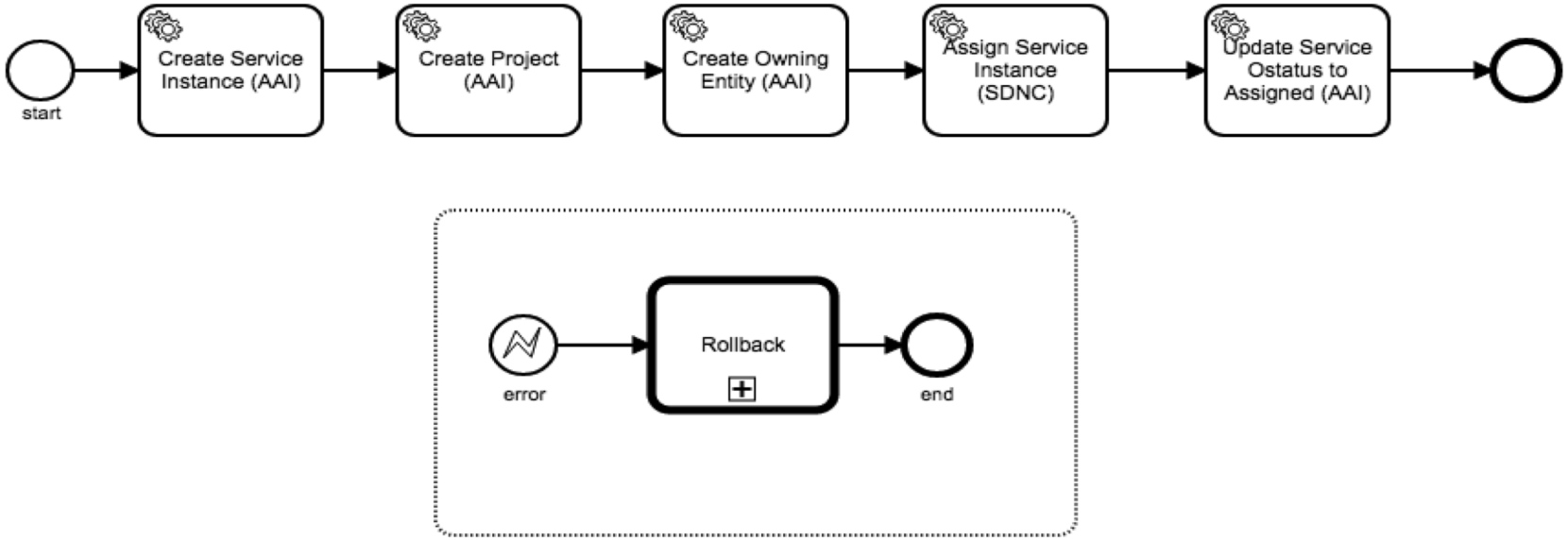


Current Service Instance BB Examples

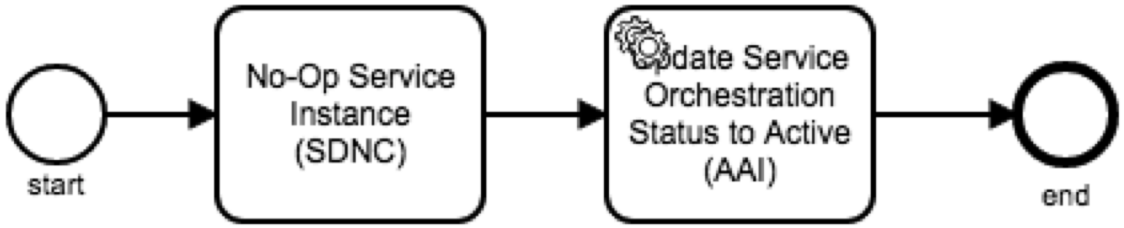


Start Process

New Assign Service Instance BB Examples

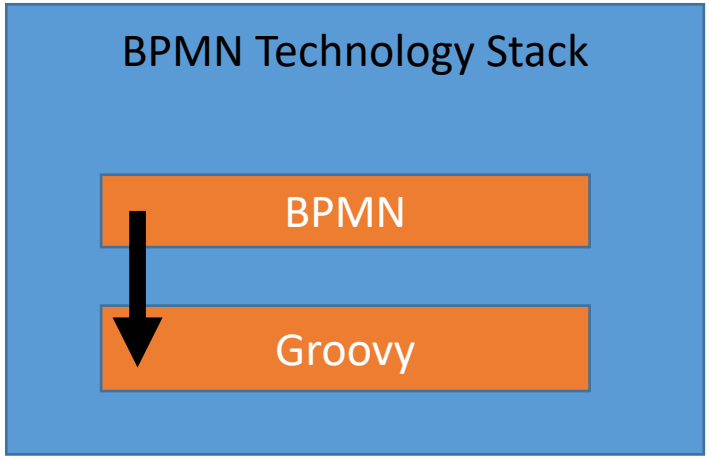


New Activate Service Instance BB Examples



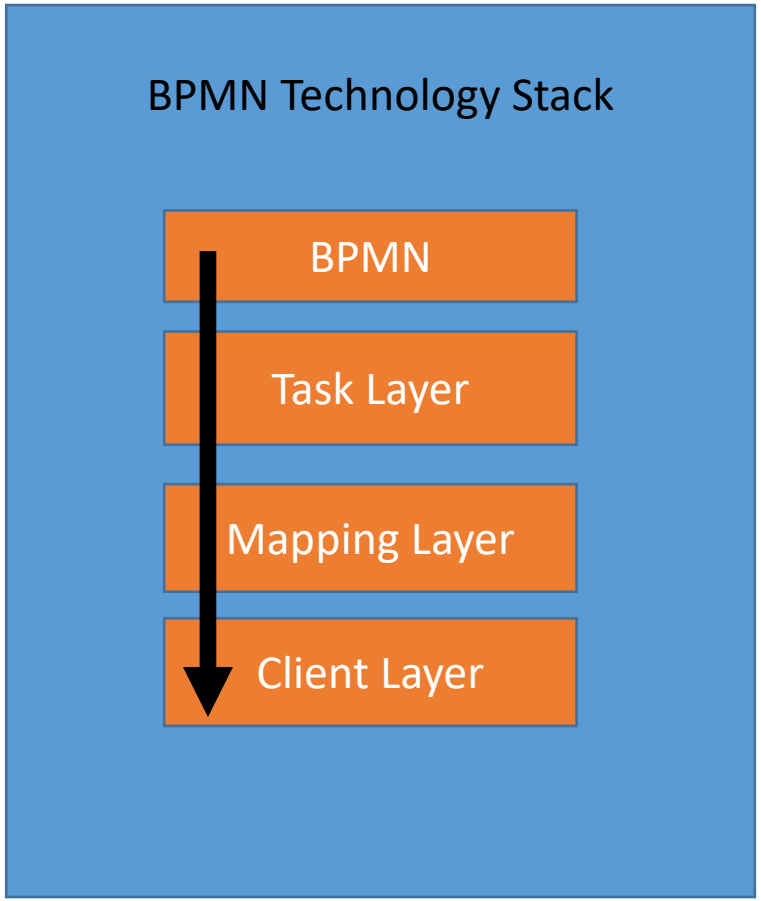
Current Task Implementation

- Groovy based
- Groovy Script per BPMN flow
- Limited re-usability between BPMN flows
- Inconsistent logging
- Difficult to test



New Layered Task Implementation

- Java based
- Easy to isolate and test
- Increased re-usability across flows
- Consistent logging mechanism
- Consistent Interface to Task Layer



Task Layer

- Provides re-usable tasks
- Interfaces using SO Domain Model
- Extracts data from Camunda context, calls mapping layer, calls clients

```
public void createServiceInstance(BuildingBlockExecution execution) {
    try {
        ServiceInstance serviceInstance = extractPojosForBB.extractByKey(execution, ResourceKey.SERVICE_INSTANCE_ID,
                execution.getLookupMap().get(ResourceKey.SERVICE_INSTANCE_ID));
        Customer customer = serviceInstance.getCustomer();
        execution.setVariable("aaiServiceInstanceRollback", false);
        String serviceInstanceId = UUID.randomUUID().toString();
        msoLogger.info("ServiceInstanceId: " + serviceInstanceId);
        serviceInstance.setServiceInstanceId(serviceInstanceId);
        aaiSIResources.createServiceInstance(serviceInstance, customer);
        execution.setVariable("aaiServiceInstanceRollback", true);
    } catch (Exception ex) {
        exceptionUtil.buildAndThrowWorkflowException(execution, 7000, ex);
    }
}
```

Mapping Layer

- Currently maps our Domain objects to the target systems interfaces
- Utilizes a java library to do the majority of heavy lifting

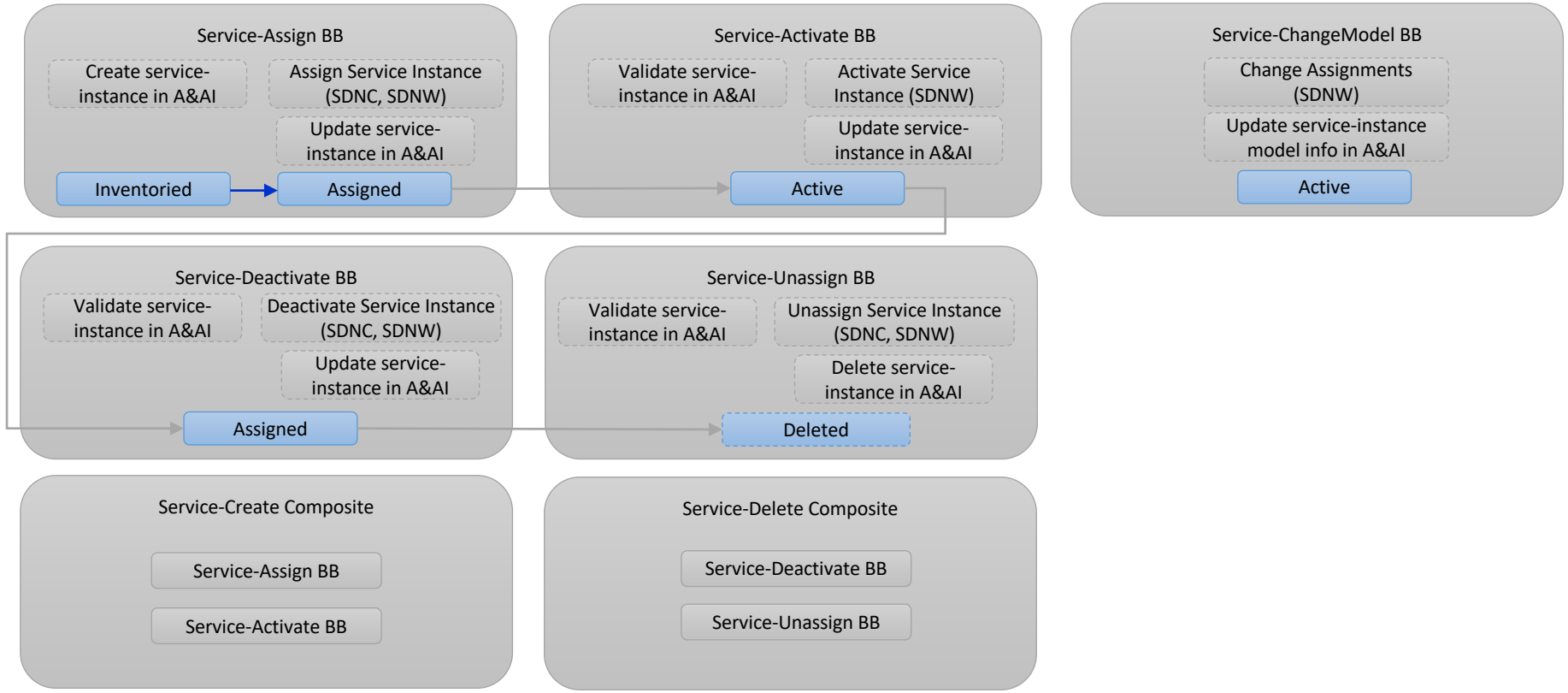
```
public org.onap.aai.domain.yang.v12.ServiceInstance mapServiceInstance (ServiceInstance serviceInstance){
    if (modelMapper.getTypeMap(ServiceInstance.class, org.onap.aai.domain.yang.v12.ServiceInstance.class) == null) {
        modelMapper.addMappings(new PropertyMap<ServiceInstance, org.onap.aai.domain.yang.v12.ServiceInstance>(){
            @Override
            protected void configure() {
                map().setServiceType(source.getModelInfoServiceInstance().getServiceType());
                map().setServiceRole(source.getModelInfoServiceInstance().getServiceRole());
                map().setModelInvariantId(source.getModelInfoServiceInstance().getModelInvariantUUID());
                map().setModelVersionId(source.getModelInfoServiceInstance().getModelUUID());
                map().setEnvironmentContext(source.getModelInfoServiceInstance().getEnvironmentContext());
                map().setWorkloadContext(source.getModelInfoServiceInstance().getWorkloadContext());
            }
        });
    }

    return modelMapper.map(serviceInstance,org.onap.aai.domain.yang.v12.ServiceInstance.class);
}
```

Client Layer

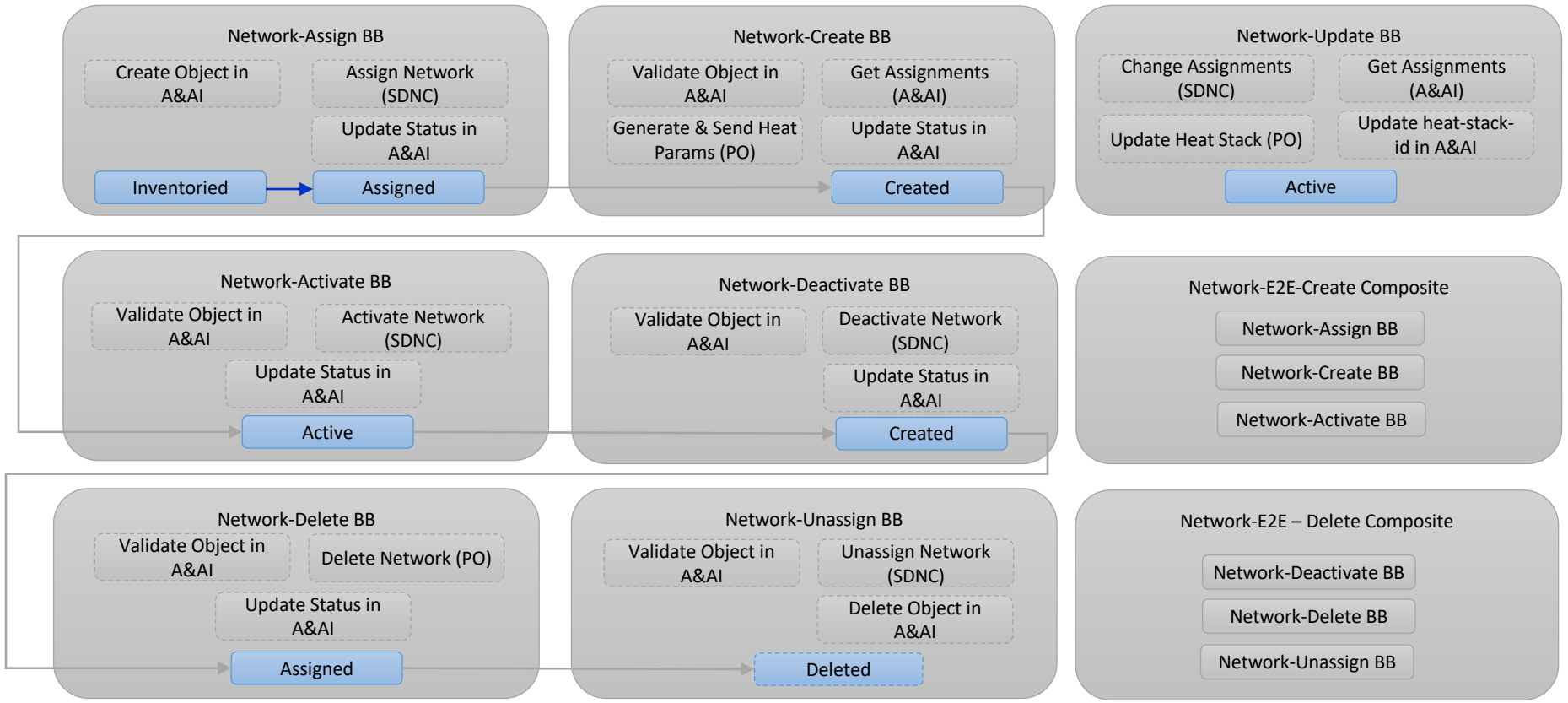
- Generic Re-usable clients to interact with other systems
- These are designed to be re-usable outside of SO
- They have no internal ties to SO in and of itself
- Reduce the work to interact with external systems
- Example Usage A&AI Client in Task Layer:

```
public void createProjectandConnectServiceInstance(Project project, ServiceInstance serviceInstance) {
    AAIResourceUri projectURI = AAIUriFactory.createResourceUri(AAIObjectType.PROJECT, project.getProjectName());
    AAIResourceUri serviceInstanceURI = AAIUriFactory.createResourceUri(AAIObjectType.SERVICE_INSTANCE,
            serviceInstance.getServiceInstanceId());
    AAIResourcesClient aaiRC = this.getClient();
    org.onap.aai.domain.yang.v12.Project AAIProject = aaiObjectMapper.mapProject(project);
    aaiRC.createIfNotExists(projectURI, Optional.of(AAIProject)).connect(projectURI, serviceInstanceURI);
}
```

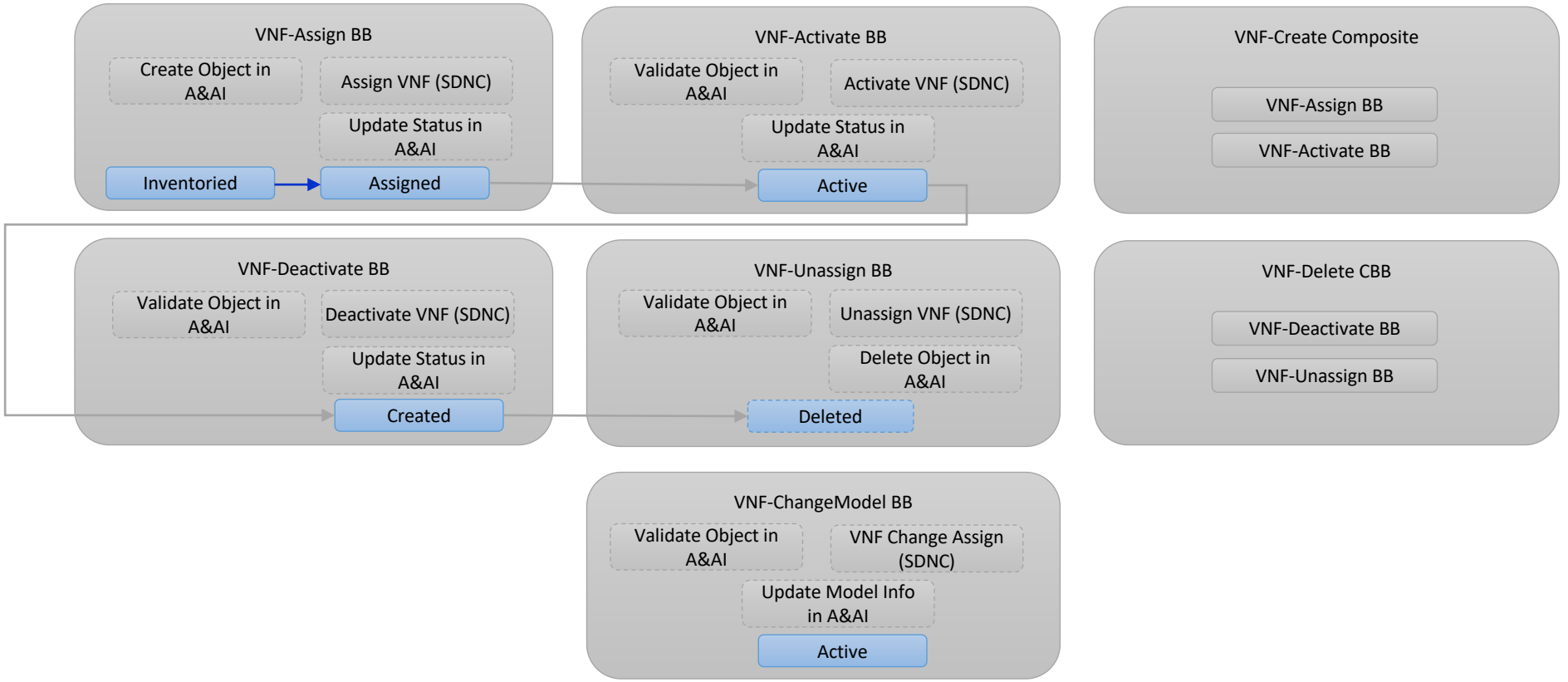
Building Blocks - Service



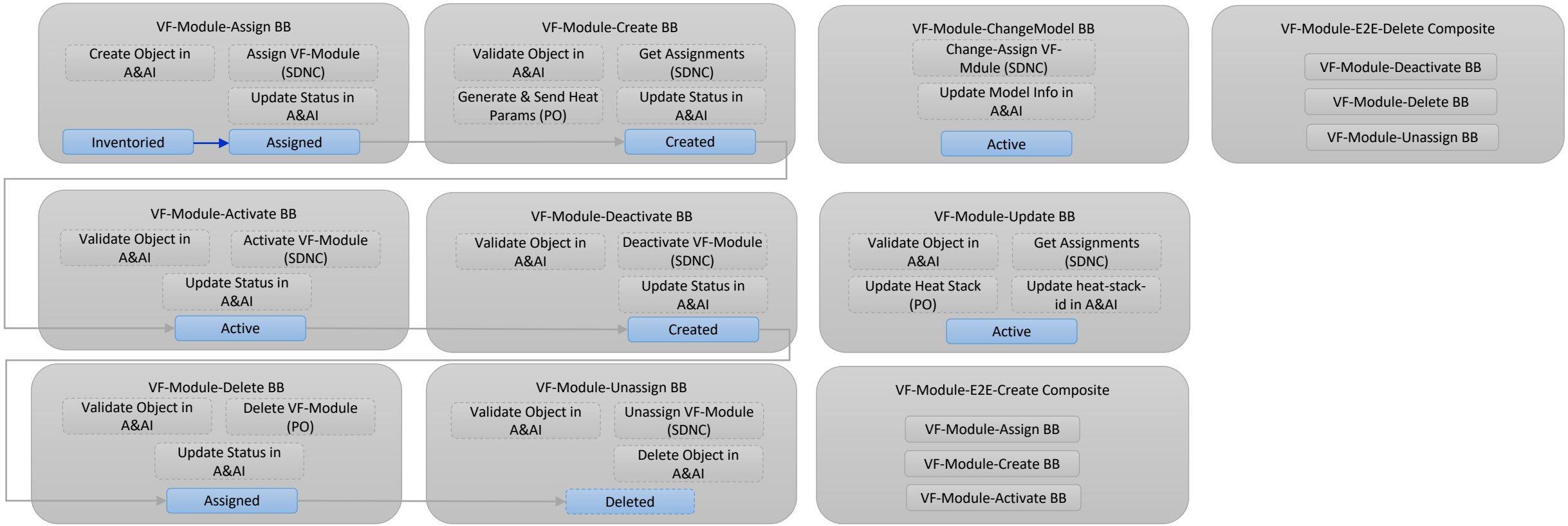
Building Blocks - Network



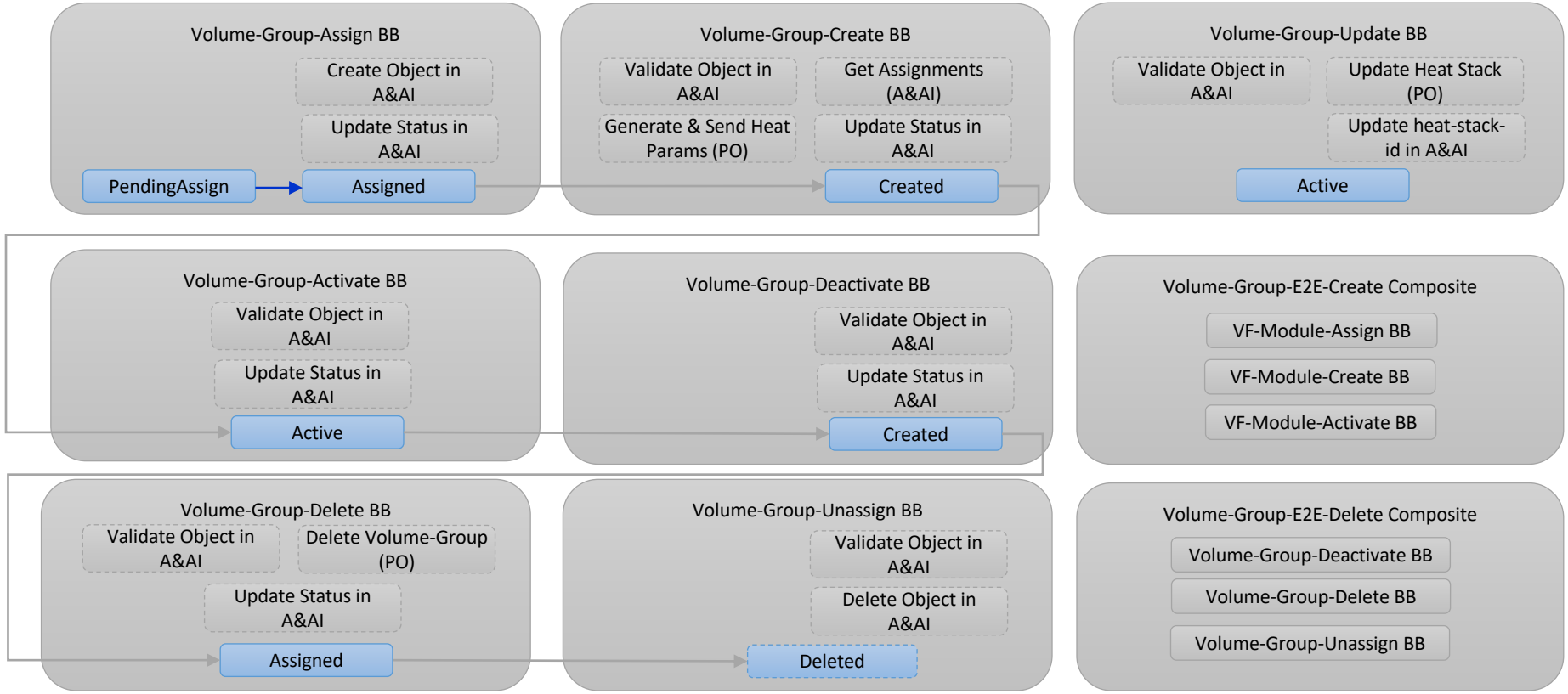
Building Blocks - VNF



Building Blocks – VF Modules



Building Blocks - Volumes



Property Management Updates

- Remove the current way urn.properties is utilized
- Migrate properties to spring based .yaml properties
- Properties are read using beans
- Properties are not injected as variables into every BPMN
- Utilize spring profiles to separate out individual environment configurations

Logging Consistency Updates

- Create JAX-RS Interceptors
  - Utilized on Service Side(audit.log)
  - Utilized for all clients to talk to other services(metrics.log)
- Migration of groovy logging to utilize the logger directly instead of utility classes
- Consistent logback.xml configuration for all services in SO
- Instead of Try Catch blocks, utilize java custom exceptions
- Moved a considerable amount of logging to trace level(Mostly entry/exit method style logging)

Unit Test and Integration Test Updates

- Remove usage of Arquillian for integration tests
- Use spring-boot Integration tests(between SO Services)
- Created additional Unit tests(class level tests)

Future Work Items

- Iterate on logging updates to target better content
- Remove shared database access between the services
- Deprecate/Remove the SOAP Interfaces and utilize REST Services
- Add Event based interfaces
- Target decomposition logic to move to its own service
- Consider changing layout/condensing bpmn projects
- Target resiliency features per service/container
  - Callbacks within the system should be more resilient

Questions and Answers