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

Docker Image: Base Ubuntu

Wildfly Web Server
- BPMN Infra WAR
- Network Adapter WAR
- SDNC Adapter WAR
- API Handler WAR
- ASDC Controller WAR
- Request DB Adapter WAR
- Catalog Adapter WAR
- VNF Adapter War
- Tenant Adapter War
- VFC Adapter
- 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
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
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

```java
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

```java
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:

```java
public void createProjectAndConnectServiceInstance(Project project, ServiceInstance serviceInstance) {
    AAISourceUri projectURI = AAISourceFactory.createResourceUri(AAIObjectType.PROJECT, project.getProjectName());
    AAISourceUri serviceInstanceURI = AAISourceFactory.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

- **Service-Assign BB**
  - Create service-instance in A&AI
  - Assign Service Instance (SDNC, SDNW)
  - Update service-instance in A&AI
  - Inventoryed → Assigned

- **Service-Activate BB**
  - Validate service-instance in A&AI
  - Activate Service Instance (SDNW)
  - Assign → Active

- **Service-Deactivate BB**
  - Validate service-instance in A&AI
  - Deactivate Service Instance (SDNC, SDNW)
  - Assigned → Deleted

- **Service-Unassign BB**
  - Validate service-instance in A&AI
  - Unassign Service Instance (SDNC, SDNW)
  - Delete service-instance in A&AI

- **Service-CREATE Composite**
  - Service-Assign BB
  - Service-Activate BB

- **Service-DELETE Composite**
  - Service-Deactivate BB
  - Service-Unassign BB

- **Service-CHANGE Model BB**
  - Change Assignments (SDNW)
  - Update service-instance model info in A&AI
Building Blocks - VNF

VNF-Assign BB
Create Object in A&AI
Assign VNF (SDNC)
Update Status in A&AI
Assigned
Inventoried

VNF-Activate BB
Validate Object in A&AI
Activate VNF (SDNC)
Update Status in A&AI
Active

VNF-Deactivate BB
Validate Object in A&AI
Deactivate VNF (SDNC)
Update Status in A&AI
Created

VNF-Unassign BB
Validate Object in A&AI
Unassign VNF (SDNC)
Delete Object in A&AI
Deleted

VNF-ChangeModel BB
Validate Object in A&AI
VNF Change Assign (SDNC)
Update Model Info in A&AI
Active

VNF-Created Composite
VNF-Assign BB
VNF-Activate BB

VNF-Delete CBB
VNF-Deactivate BB
VNF-Unassign BB
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