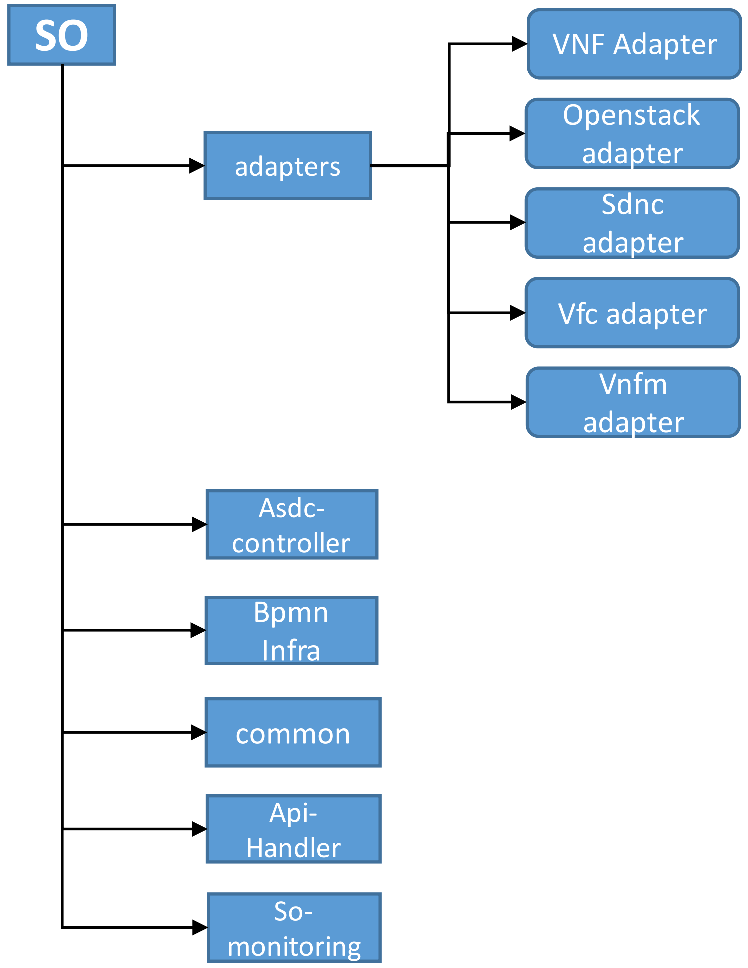
# SO adapter development with south bound P&P

Prepared by : Isaac, Seshu

# SO Project Structure



SO has the following core projects, the responsibility of the projects are mentioned below:

## ➢Adapters:

Provided Interfaces to lower level controllers and other ONAP components

i. Catalog-db-adapter: interfaces to query the service/resource recipe and service catalog info.
ii. Request-db-adapter: interfaces to query/update the service request information.
iii. Openstack-adapter: support Create, Instantiate, Terminate/Delete over openstack APIs.
iv. Sdnc-adapter: interfaces to create network resources and interact with other controllers.
v. Vfc-adapter:
vi. Vnfm-adapter: support Create, Instantiate, Terminate/Delete on SOL003 based VNF client.

## ➢Asdc controller:

i. Receive updated service models from SDCEvent-bus notifications when new models available.
ii. HTTP retrieval of models (TOSCA) and artifacts (Heat)
iii. Receive distributions as TOSCA models
iv. Populate SO Catalog

## ➢Bpmn Infra

i. Sequence orchestration steps for each Resource in the recipe
ii. Request and configure network resources via SDN-C

## ➢Common

Holds the beans and utils common for all the projects.

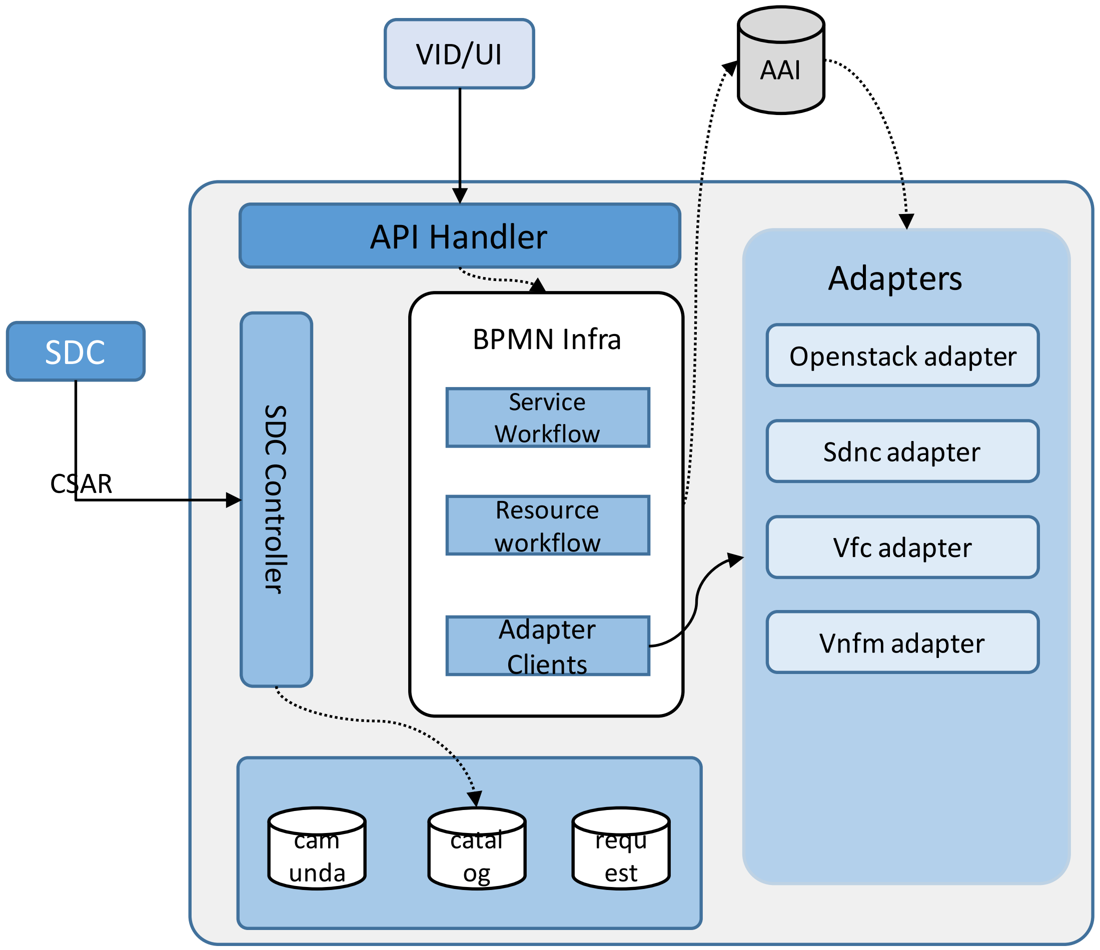
## ➢Api Handler

i. Provides REST API for northbound clients. Responsible to handle the service and infrastructure level request, like service instantiation.
ii. Provides model driven recipe selection, using catalog DB.
iii. Forwards the request to the bpmn recipe selected based on service-model + action (dynamic look up).

## ➢So monitoring

i. Monitor BPMN Workflow execution by providing search criteria

# SO Architecture insight



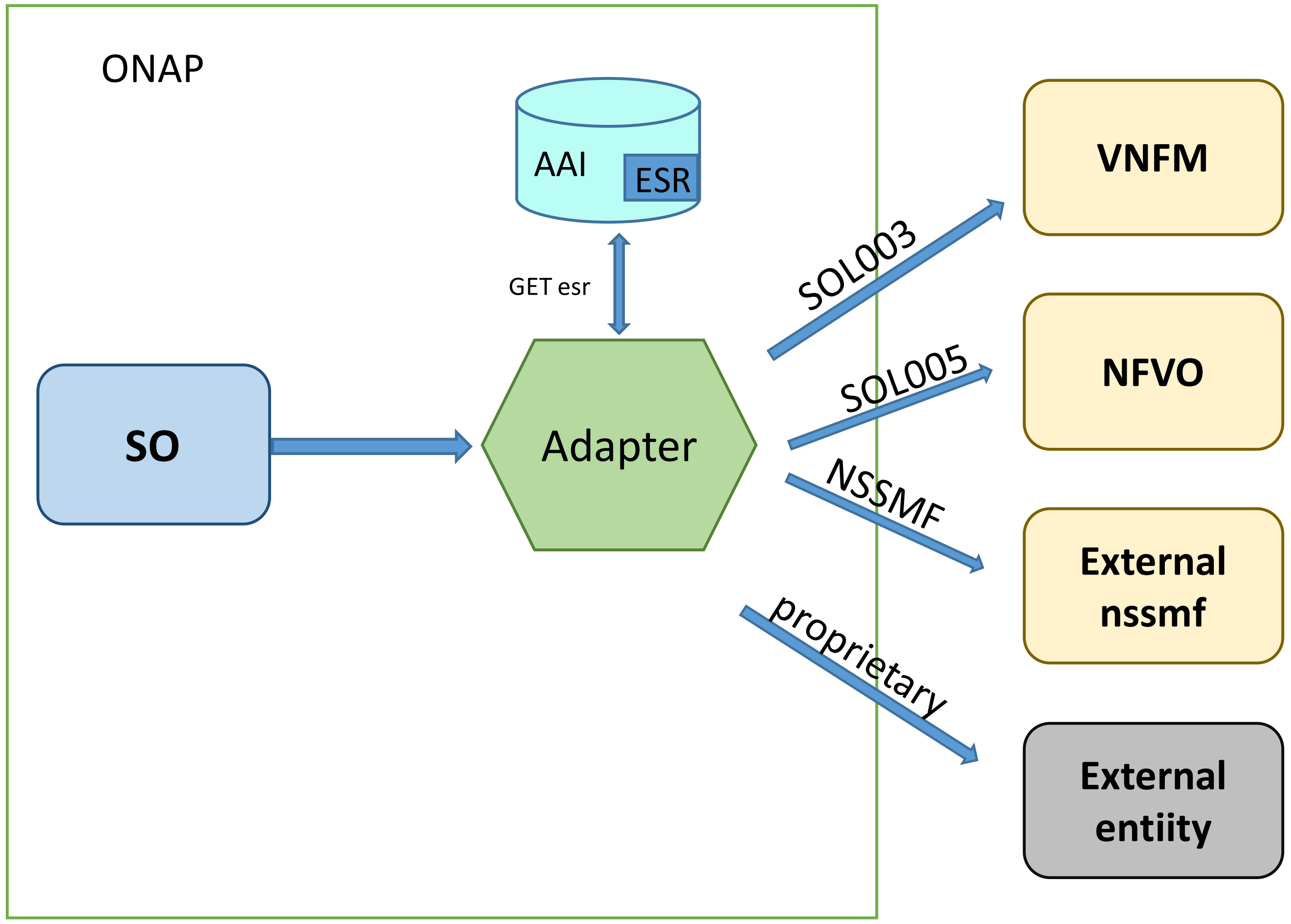
Stage 1:

- SDC Distribute the service archive.
- Sdc controller parse the archive and populate the catalog db.

Stage 2:

- VID or usecase UI send request to BPMN Infra.
- BPMN infra fetch the right recipe from catalog DB and orchestrate the service and resource creation.
- Adapter client will be used to communicate with adapters.

# How to make a plug & play adapter



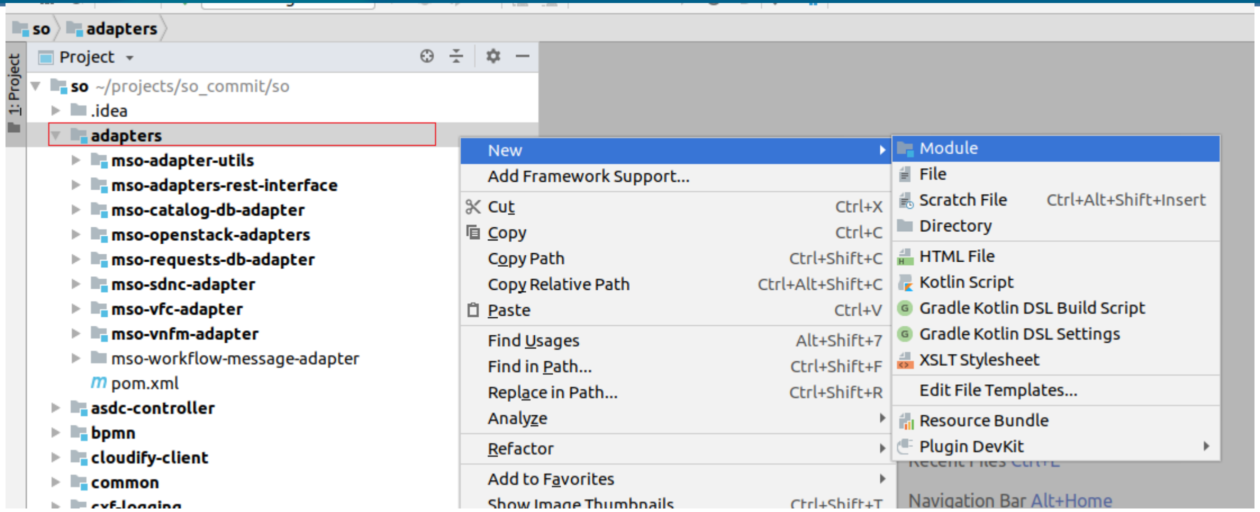
To have a plug & play adapter, the following rule has to be followed during implementation:

- ✓ South bound interface must implement based on any standard.
- ✓ All those standard must be supported by the adapter.
- ✓ All plug & play south bound entities should implement same functionality.
- ✓ when service resource is designed, the vendor and node information needs to be added along with the model information.
- ✓ The ESR information in AAI also needs to be preloaded, matching the design time parameters.
- ✓ When adapter sends the request to south bound nodes, query AAI and use the ESR information to send request.

# Create New Adapter (nssmf adapter)

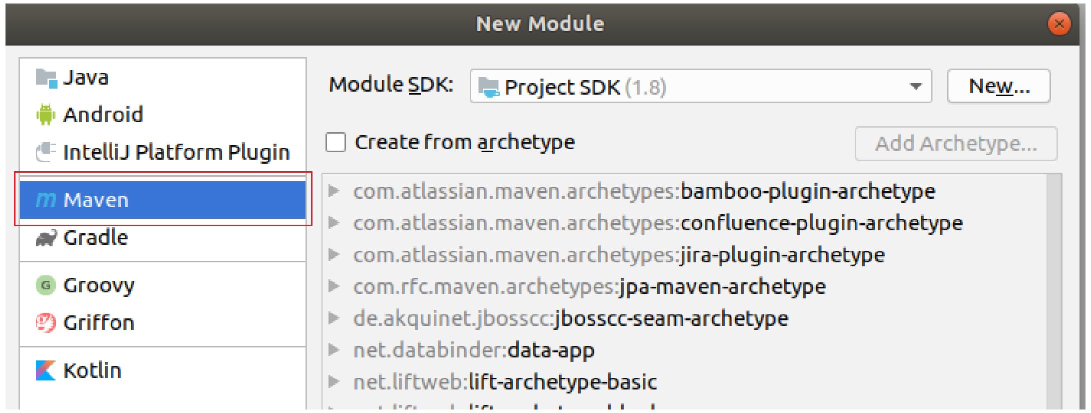
Create a new module under adapters.

1. Select “New” Option in the pop-up menu.
2. Select “Module” option in the sub menu.



Add Maven support to it.

You can add maven support to the project. This will add the pom file to the poject. Later you will just need to add the dependencies.

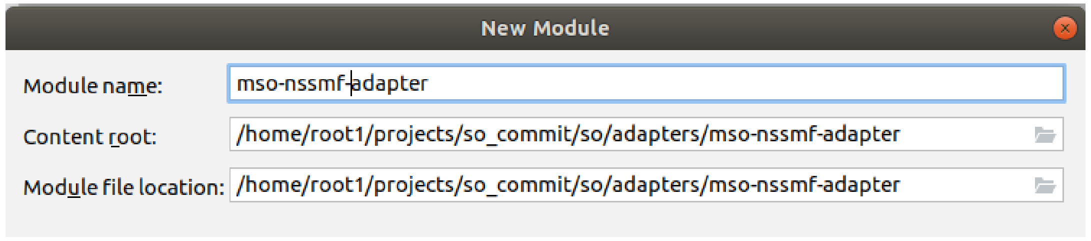


New Module
Java
Android
IntelliJ Platform Plugin
Maven
Gradle
Groovy
Griffon
Kotlin
Module SDK: Project SDK (1.8)
New...
Create from archetype
Add Archetype...
com.atlassian.maven.archetypes:bamboo-plugin-archetype
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic

# Create New Adapter

Enter the module name.

1. You will need to enter the module name to be created.
2. Make sure the path is rightly populated before proceeding.

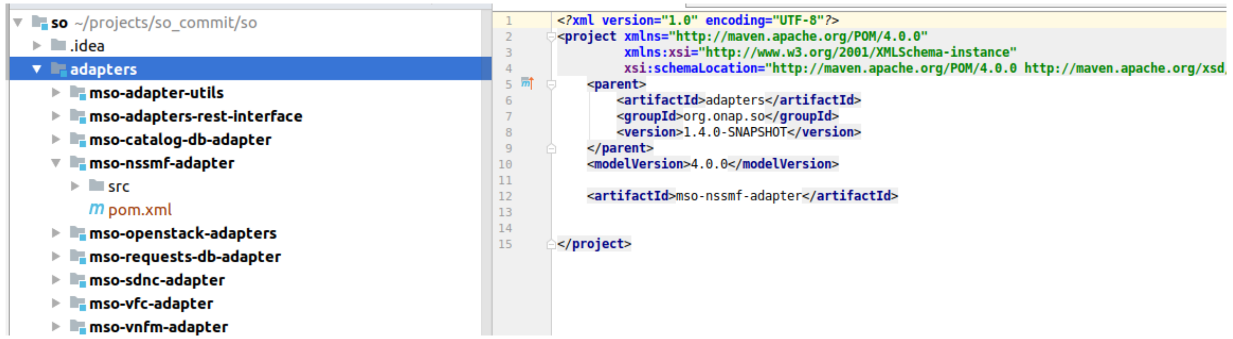


New Module

| | |
|---|---|
| Module name: | mso-nssmf-adapter |
| Content root: | /home/root1/projects/so_commit/so/adapters/mso-nssmf-adapter |
| Module file location: | /home/root1/projects/so_commit/so/adapters/mso-nssmf-adapter |

Once successfully created you could see the pom file of the module

1. Make sure that the parent project information is populated correctly.
2. Make sure the artifactid for the new adapter is generated correctly



```
so ~/projects/so_commit/so
  .idea
  adapters
    mso-adapter-utils
    mso-adapters-rest-interface
    mso-catalog-db-adapter
    mso-nssmf-adapter
      src
      pom.xml
    mso-openstack-adapters
    mso-requests-db-adapter
    mso-sdnc-adapter
    mso-vfc-adapter
    mso-vnfm-adapter
```

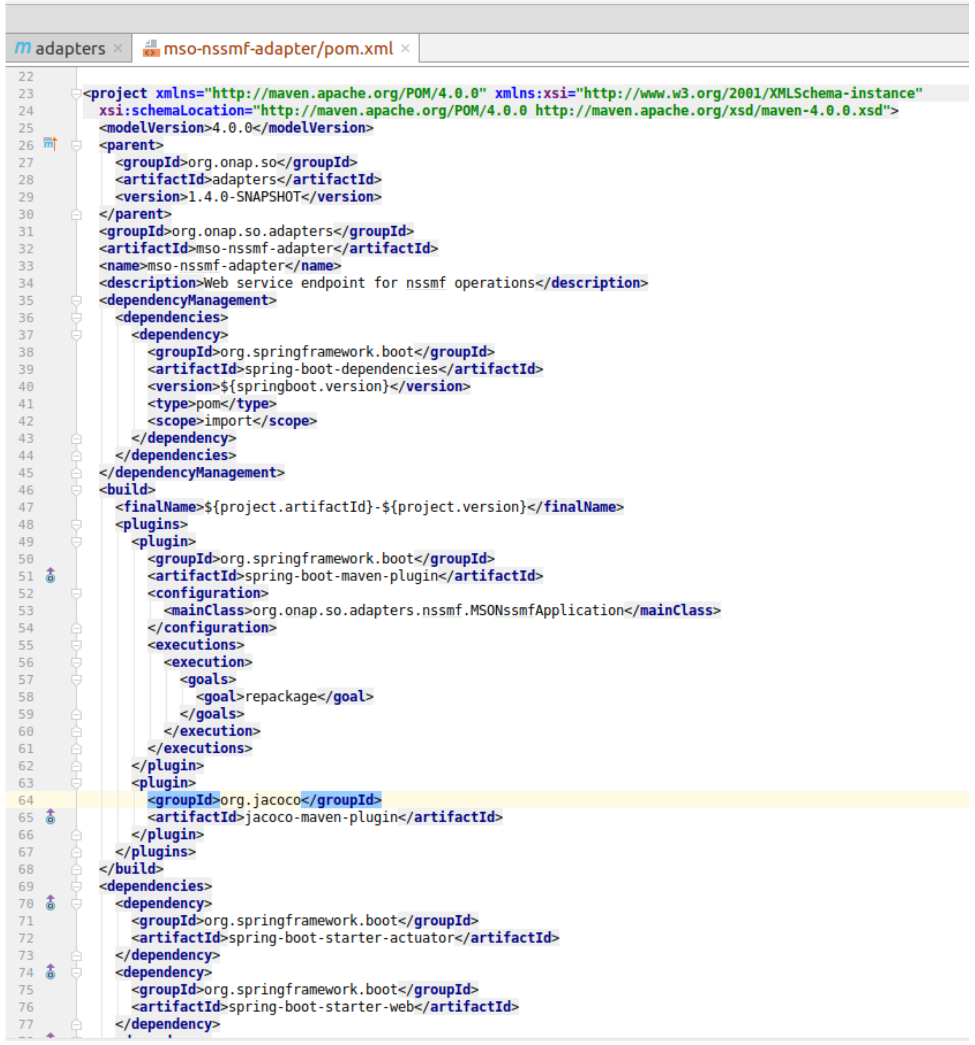
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd,
    <parent>
        <artifactId>adapters</artifactId>
        <groupId>org.onap.so</groupId>
        <version>1.4.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mso-nssmf-adapter</artifactId>


</project>
```

# Create New Adapter

Once the project is created, need to add required dependencies to the pom file of the new module.

1. Add all the required dependencies. Make sure following dependencies are added:
   1. spring boot dependencies.
   2. Build goals

Tips: You can copy dependencies and pom build configurations from other adapter projects.



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.onap.so</groupId>
    <artifactId>adapters</artifactId>
    <version>1.4.0-SNAPSHOT</version>
  </parent>
  <groupId>org.onap.so.adapters</groupId>
  <artifactId>mso-nssmf-adapter</artifactId>
  <name>mso-nssmf-adapter</name>
  <description>Web service endpoint for nssmf operations</description>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>${springboot.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <build>
    <finalName>${project.artifactId}-${project.version}</finalName>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <mainClass>org.onap.so.adapters.nssmf.MSONssmfApplication</mainClass>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>repackage</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

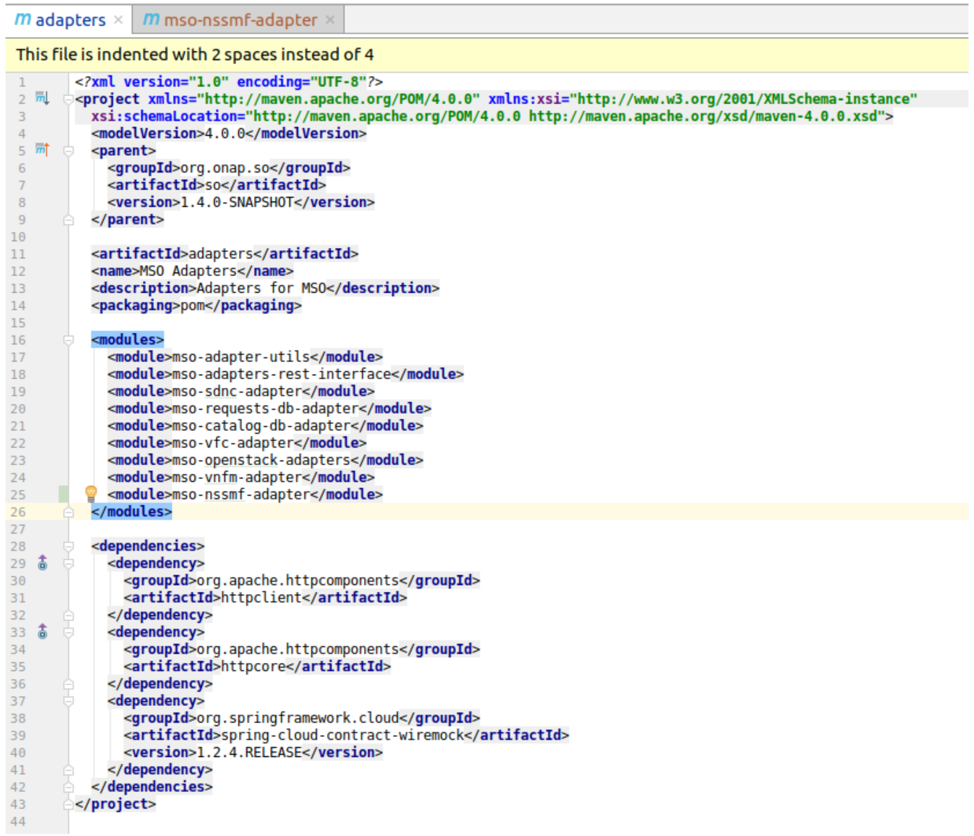
# Create New Adapter

Once the module is created successfully,

1. Need to add the module to the adapter project.
2. Go to the the pom file of adapters project.
3. Add the newly created module in the modules tag.

Now the newly created adapter will be part of the adapters project.

When adapters project is complied the new adapter module will also be compiled and built.



This file is indented with 2 spaces instead of 4

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.onap.so</groupId>
    <artifactId>so</artifactId>
    <version>1.4.0-SNAPSHOT</version>
  </parent>

  <artifactId>adapters</artifactId>
  <name>MSO Adapters</name>
  <description>Adapters for MSO</description>
  <packaging>pom</packaging>

  <modules>
    <module>mso-adapter-utils</module>
    <module>mso-adapters-rest-interface</module>
    <module>mso-sdnc-adapter</module>
    <module>mso-requests-db-adapter</module>
    <module>mso-catalog-db-adapter</module>
    <module>mso-vfc-adapter</module>
    <module>mso-openstack-adapters</module>
    <module>mso-vnfm-adapter</module>
    <module>mso-nssmf-adapter</module>
  </modules>

  <dependencies>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
    </dependency>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpcore</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-contract-wiremock</artifactId>
      <version>1.2.4.RELEASE</version>
    </dependency>
  </dependencies>
</project>
```

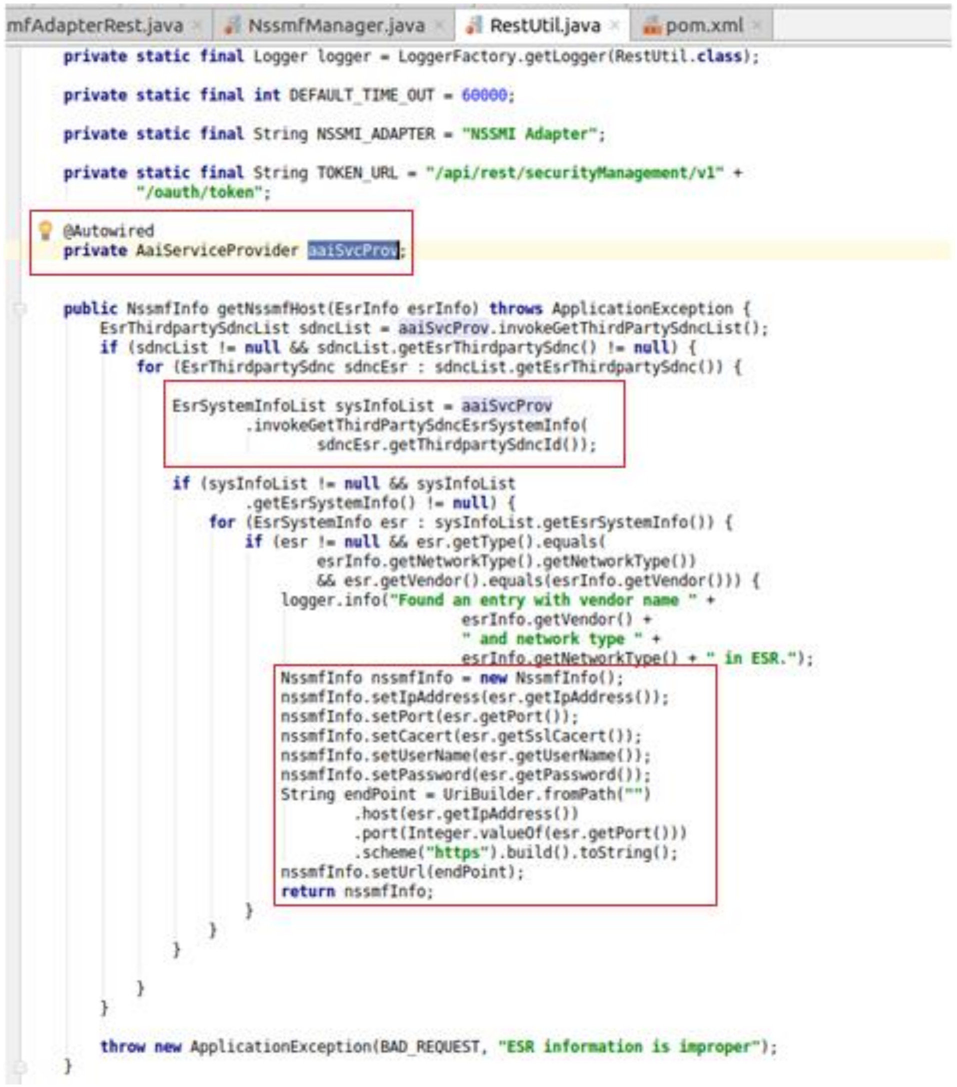
# How to make a plug & play adapter

While connecting to the south bound node,

1. Get the node information from AAI ESR information.
2. Parse the ESR information and get the IP and Port.
3. Prepare the end point using the ESR information
4. Fire the request to the end point.

Note:

1. ESR information should be a day zero configuration.
2. The ESR information should match with the service design.



```java
    private static final Logger logger = LoggerFactory.getLogger(RestUtil.class);

    private static final int DEFAULT_TIME_OUT = 60000;

    private static final String NSSMI_ADAPTER = "NSSMI Adapter";

    private static final String TOKEN_URL = "/api/rest/securityManagement/v1" +
            "/oauth/token";

    @Autowired
    private AaiServiceProvider aaiSvcProv;

    public NssmfInfo getNssmfHost(EsrInfo esrInfo) throws ApplicationException {
        EsrThirdpartySdncList sdncList = aaiSvcProv.invokeGetThirdPartySdncList();
        if (sdncList != null && sdncList.getEsrThirdpartySdnc() != null) {
            for (EsrThirdpartySdnc sdncEsr : sdncList.getEsrThirdpartySdnc()) {

                EsrSystemInfoList sysInfoList = aaiSvcProv
                        .invokeGetThirdPartySdncEsrSystemInfo(
                                sdncEsr.getThirdpartySdncId());

                if (sysInfoList != null && sysInfoList
                        .getEsrSystemInfo() != null) {
                    for (EsrSystemInfo esr : sysInfoList.getEsrSystemInfo()) {
                        if (esr != null && esr.getType().equals(
                                esrInfo.getNetworkType().getNetworkType())
                                && esr.getVendor().equals(esrInfo.getVendor())) {
                            logger.info("Found an entry with vendor name " +
                                                esrInfo.getVendor() +
                                                " and network type " +
                                                esrInfo.getNetworkType() + " in ESR.");
                            NssmfInfo nssmfInfo = new NssmfInfo();
                            nssmfInfo.setIpAddress(esr.getIpAddress());
                            nssmfInfo.setPort(esr.getPort());
                            nssmfInfo.setCacert(esr.getSslCacert());
                            nssmfInfo.setUserName(esr.getUserName());
                            nssmfInfo.setPassword(esr.getPassword());
                            String endPoint = UriBuilder.fromPath("")
                                    .host(esr.getIpAddress())
                                    .port(Integer.valueOf(esr.getPort()))
                                    .scheme("https").build().toString();
                            nssmfInfo.setUrl(endPoint);
                            return nssmfInfo;
                        }
                    }
                }

            }
        }

        throw new ApplicationException(BAD_REQUEST, "ESR information is improper");
    }
```

## THANKS