ONAP Controllers – developing new LCM API

Note! This tutorial is based on LCM API. For other types of APIs the process may be slightly different.

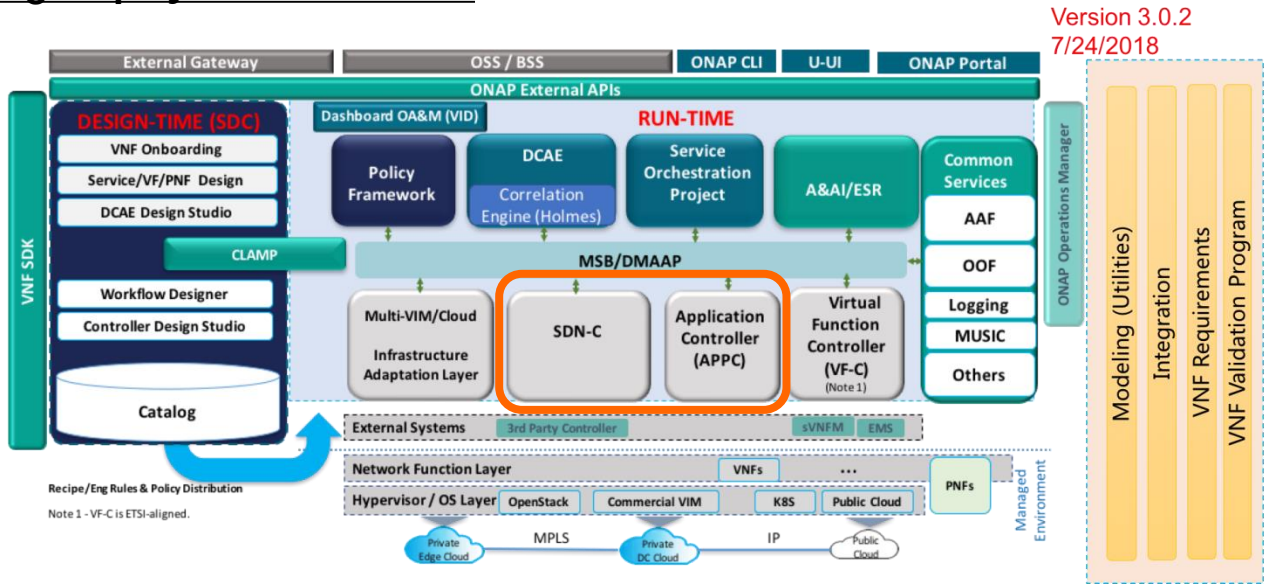
Tomek Osiński
Łukasz Rajewski

orange™

08.08.2018

Introduction

- **ONAP uses different Controller types:**
  - **SDN-C – responsible for carrying out network configuration**
  - **APP-C – manages application-level lifecycle management (LCM)**
- **More info: https://wiki.onap.org/display/DW/Controllers**

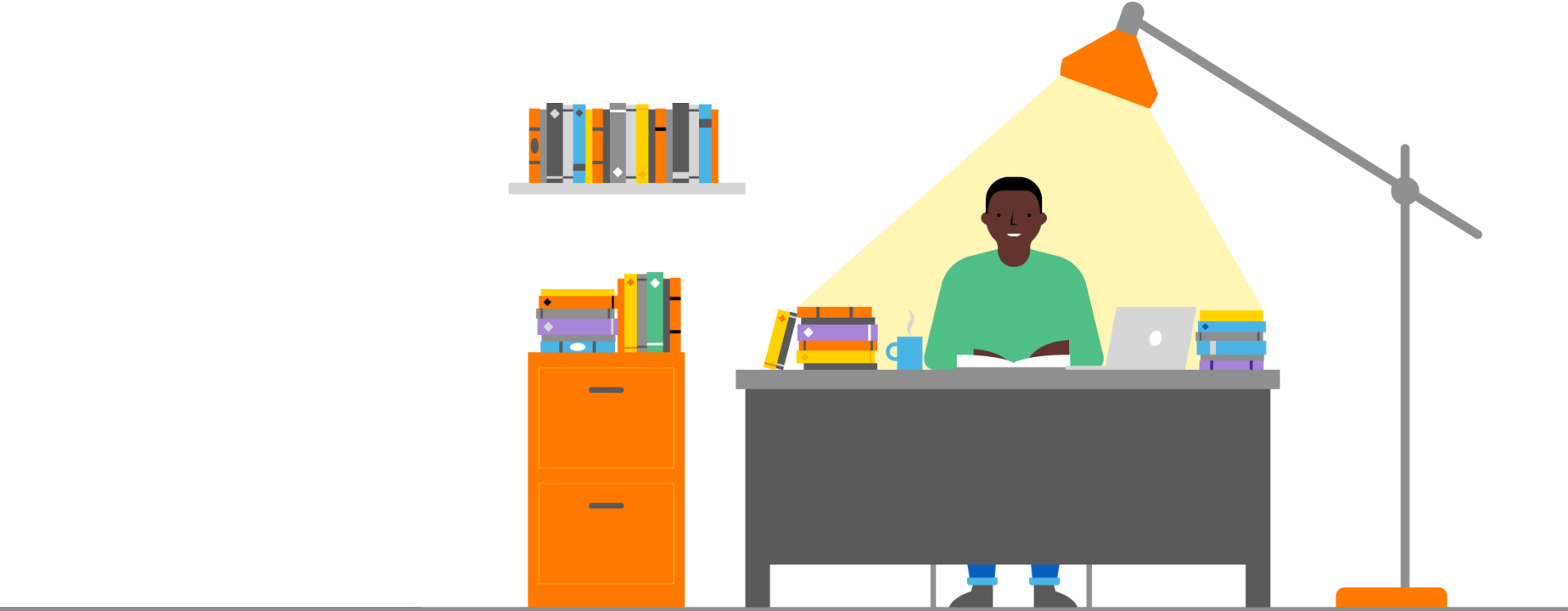


Key concepts (1/2)

- Both ONAP Controllers are based on OpenDaylight
- Opendaylight
  - SDN controller implemented based on OSGi and Karaf
  - supports model-driven service abstraction layer (MD-SAL)
- YANG
  - Modeling language for config data, state data, RPCs and notifications
  - YANG is used in OpenDaylight to generate data models for RPCs (Northbound APIs) and data store
  - More info: https://www.slideshare.net/GunjanPatel4/yang-in-odl

Key concepts (2/2)

- The Service Logic for ONAP Controllers is defined by Directed Graphs (DGs)
- The common code for SDN-C/APP-C is located in CCSDK (Common Controller SDK)
- Directed Graph
  - DG defines the pipeline of actions to be invoked for particular RPC method
  - In fact, DG implements the service logic for Northbound APIs of ONAP Controllers
  - More info: https://wiki.onap.org/display/DW/Service+Logic+Interpreter+Directed+Graph+Guide
- CCSDK
  - As SDN-C and APP-C have a lot of commonalities there has been made a decision to migrate a common code to SDK library
  - CCSDK provides a common set of reusable code that can be used across multiple controllers (not only in APP-C or SDN-C)



SDN-C

Project structure

- SDN-C is strongly based on CCSDK. It inherits a definition of northbound APIs from CCSDK.
- The SDN-C related code is located in SDN-C repository
- Important modules:
  - ccsdk/sli/northbound – defines northbound interface (e.g. DMaaP consumer, LCM APIs)
  - ccsdk/distribution – contains files to build CCSDK Docker images (dgbuilder, ccsdk-odlsli, ccsdk-odl-oxygen, ccsdk-ubuntu)
  - sdnc/northbound – defines northbound interface for SDN-C (e.g. VNF API, Generic Resource API)
  - sdnc/oam – contains files to build SDN-C Docker images (sdnc-ueb-listener, sdnc-dmaap-listener, sdnc-ansible-server, admportal-sdnc, sdnc)
  - sdnc/oam/platform-logic – contains Directed Graphs implementing service logic for northbound APIs

Implementing new Northbound API for SDN-C

- This tutorial is based on our expierience from implementing LCM Traffic Migration API for ONAP Change Management purposes

- Steps to implement new LCM API for SDN-C:
  - Extend YANG model in CCSDK
  - Compilation of YANG models in CCSDK
  - Implement new method in LcmProvider.java in CCSDK
  - Implement Directed Graph with service logic in SDN-C
  - Configure SDN-C to activate DG at startup

Extend YANG model in CCSDK

ccsdk/sli/northbound/lcm/model/src/main/yang/lcm.yang

- **In lcm.yang extend data model with new RPC method:**
  1. **Add new enum (e.g. MigrateTraffic). This new Enum will define an Action in LCM API.**
  2. **Define new RPC Method (e.g. migrate-traffic)**
- **Note: This method contains standard input parameters (common-header, action, action-identfiers and payload). Thanks to that you don't need to modify header parsing in Java code. You can put you custom parameters to content of payload.**

lcm/model/src/main/yang/lcm.yang

```
...  ...  @@ -121,6 +121,7 @@ module LCM {
121  121           enum "Reboot";
122  122           enum "AttachVolume";
123  123           enum "DetachVolume";
     124  +        enum "MigrateTraffic";
124  125
125  126       }
126  127       description "The action to be taken by APP-C, e.g. Restart, Rebuild, Migrate";
...  ...  @@ -1301,6 +1302,29 @@ module LCM {
1301 1302       }
1302 1303   }
1303 1304
     1305 +  /*****************************************************************************
     1306 +   * Define the traffic migration service
     1307 +   *****************************************************************************/
     1308 +  rpc migrate-traffic {
     1309 +      description "An operation to migrate traffic from a VM";
     1310 +      input {
     1311 +          uses common-header;
     1312 +          leaf action {
     1313 +              type action;
     1314 +              mandatory true;
     1315 +          }
     1316 +          uses action-identifiers;
     1317 +          leaf payload {
     1318 +              type payload;
     1319 +              mandatory true;
     1320 +          }
     1321 +      }
     1322 +      output {
     1323 +          uses common-header;
     1324 +          uses status;
     1325 +      }
     1326 +  }
```

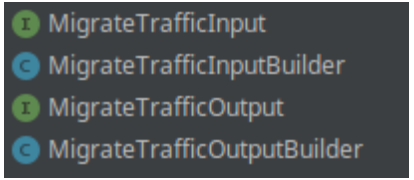
Compilation of YANG models in CCSDK

- You need to compile ccsdk/sli/northbound now in order to generate Java classes based on YANG models
- Use: mvn clean install
- Maven will generate following classes (--->)
- Note! The compilation will fail, but you will see new classes generated

ccsdk/sli/northbound/lcm/model/src/main/yang-gen-sal/org/opendaylight/yang/gen/v1/org/onap/ccsdk/sli/northbound/lcm/rev180329/LCMService.java

```
/**
 * An operation to migrate traffic from a VM
 *
 */
@CheckReturnValue
Future<RpcResult<MigrateTrafficOutput>> migrateTraffic(MigrateTrafficInput input);
```

ccsdk/sli/northbound/lcm/model/src/main/yang-gen-sal/org/opendaylight/yang/gen/v1/org/onap/ccsdk/sli/northbound/lcm/rev180329




```
public interface MigrateTrafficOutput
    extends
    CommonHeader,
    Status,
    DataObject,
    Augmentable<org.opendaylight.yang.gen.v1.org.onap.ccsdk.sli.northbound.lcm.rev180329.MigrateTrafficOutput>
```

```
public interface MigrateTrafficInput
    extends
    CommonHeader,
    ActionIdentifiers,
    DataObject,
    Augmentable<org.opendaylight.yang.gen.v1.org.onap.ccsdk.sli.northbound.lcm.rev180329.MigrateTrafficInput>
```

Implement new method in LcmProvider.java in CCSDK

- Implement new method from LCMService.java interface – example below (migrateTraffic)
- Note! The most important is to call appropriate DG with RPC method (migrate-traffic in this example)

ccsdk/sli/northbound/lcm/provider/src/main/java/org/onap/ccsdk/sli/northbound/LcmProvider.java

```
@Override
public Future<RpcResult<MigrateTrafficOutput>> migrateTraffic(MigrateTrafficInput input) {
    MigrateTrafficInputBuilder inputBuilder = new MigrateTrafficInputBuilder(input);
    MigrateTrafficOutputBuilder outputBuilder = new MigrateTrafficOutputBuilder();

    try {
        CommonLcmFields retVal = callDG( rpcName: "migrate-traffic", inputBuilder.build());
    } catch (LcmRpcInvocationException e) {
        LOG.debug("Caught exception", e);
        outputBuilder.setCommonHeader(e.getCommonHeader());
        outputBuilder.setStatus(e.getStatus());
    }

    RpcResult<MigrateTrafficOutput> rpcResult =
            RpcResultBuilder.<~> status(true).withResult(outputBuilder.build()).build();

    return Futures.immediateFuture(rpcResult);
}
```

Implement Directed Graph with service logic in SDN-C (1/2)

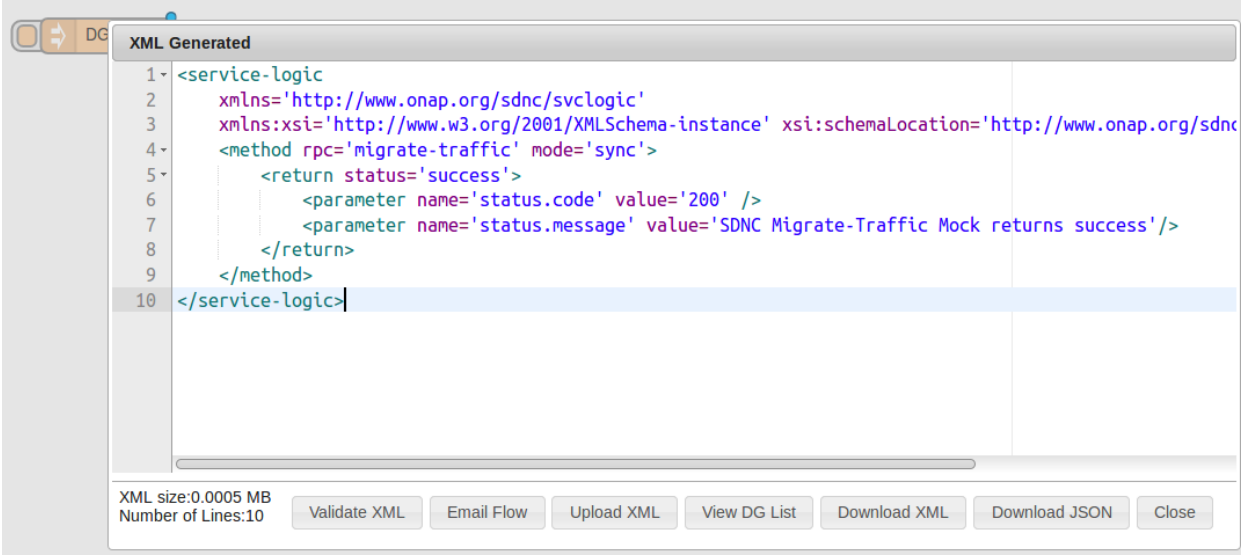
- The best way to generate Directed Graph is to use dgbuilder
- You can find a step-by-step tutorial how to make you first DG here: https://wiki.onap.org/display/DW/Your+First+Graph
- When DG is ready download both JSON and XML files
- Copy JSON file to proper dir (example for LCM API):

sdnc/oam/platform-logic/lcm/src/main/json

- Copy XML file to:

sdnc/oam/platform-logic/lcm/src/main/xml

View from dgbuilder GUI:



Configure SDN-C to activate DG at startup

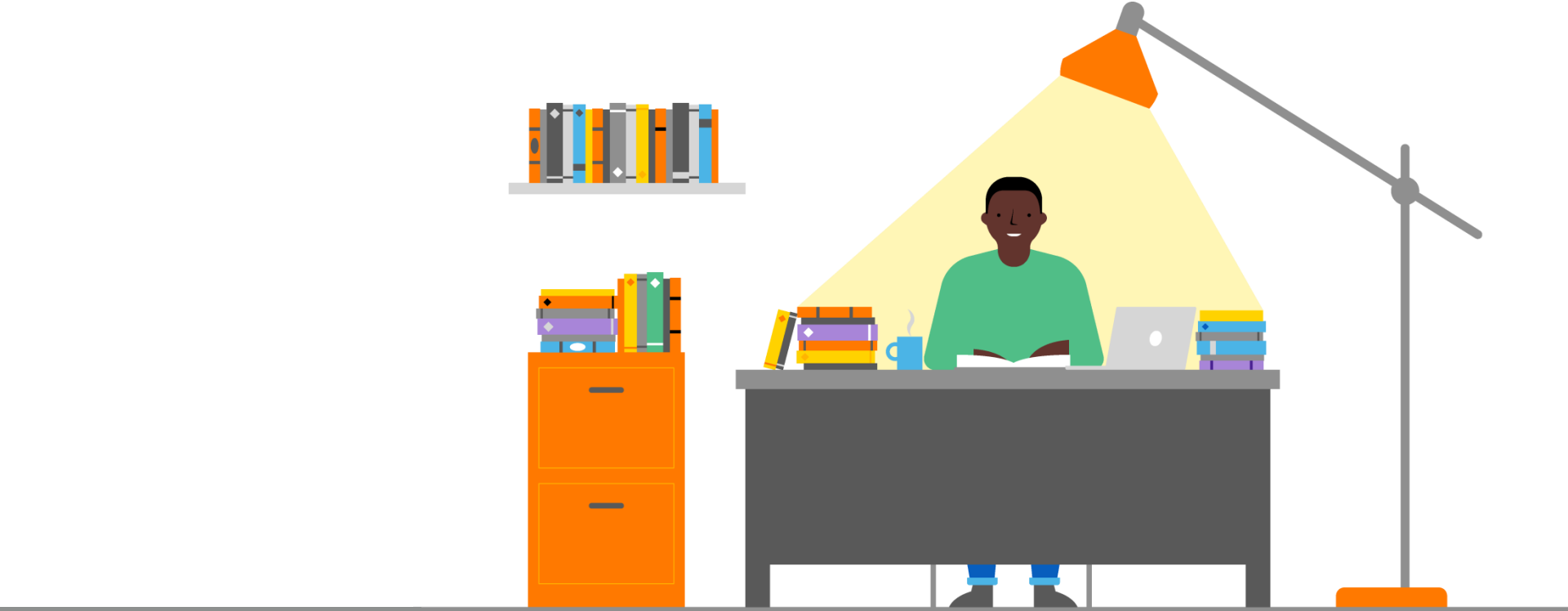
- DG must be activated to use it
- You can do it manually via dgbuilder UI
- The best way is to configure SDN-C to build along with Docker container
  - Modify sdnc/oam/platform-logic/lcm/src/main/resources/graph.versions file and append line pointing to new DG
    - Module: LCM
    - RPC name: migrate-traffic

platform-logic/lcm/src/main/resources/graph.versions

```
1  1   LCM upgrade-software ${project.version} sync
2  2   LCM upgrade-post-check ${project.version} sync
3  3   LCM upgrade-pre-check ${project.version} sync
   4 + LCM migrate-traffic ${project.version} sync
```

Installation and deployment

- Compile CCSDK and SDN-C projects
- Deploy CCSDK Docker containers
- Deploy SDN-C Docker containers



APP-C

Project structure

- For APPC Northbound APIs are implemented using similar concepts, but the project structure is different
- In opposite to SDN-C that uses code from CCSDK to implement LCM APIs, APPC uses its local API implementation
- Important modules:
  - appc/appc-provider/appc-provider-model – contains defintion of YANG models
  - appc/appc-provider/appc-provider-bundle – contains implementation of Services for handling Northbound APIs
  - appc/appc-directed-graph/appc-dgraph – contains Directed Graphs
  - appc/deployment – contains files to build Docker containers

Implementing new Northbound API for APPC

- This tutorial is based on our expierience from implementing LCM Traffic Migration API for ONAP Change Management purposes

- Steps to implement new LCM API for APP-C:
  - Extend YANG model in APPC (appc-provider-model)
  - Compilation of YANG models in APPC
  - Implement new Service to handle RPC method (e.g. MigrateTrafficService) (appc-provider-bundle)
  - Implement new method in AppcProviderLcm.java (appc-provider-bundle)
  - Add RPC method to VnfOperation list
  - Implement Directed Graph with service logic in APPC (appc-directed-graph/appc-dgraph)
  - Configure APPC to activate DG at startup (appc-directed-graph/appc-dgraph)

Extend YANG model in APPC

appc/appc-provider/appc-provider-model/src/main/yang/appc-provider-lcm.yang

- **In appc-provider-lcm.yang extend data model with new RPC method:**
  1. **Add new enum (e.g. MigrateTraffic). This new Enum will define an Action in LCM API.**
  2. **Define new RPC Method (e.g. migrate-traffic)**
- **Note: This method contains standard input parameters (common-header, action, action-identfiers and payload). Thanks to that you don't need to modify header parsing in Java code. You can put you custom parameters to content of payload.**

appc-provider/appc-provider-model/src/main/yang/appc-provider-lcm.yang

```
...  ...  @@ -121,6 +121,7 @@ module appc-provider-lcm {
121  121           enum "Reboot";
122  122           enum "AttachVolume";
123  123           enum "DetachVolume";
     124  +        enum "MigrateTraffic";
124  125
125  126       }
126  127       description "The action to be taken by APP-C, e.g. Restart, Rebuild, Migrate";
...  ...  @@ -1314,6 +1315,29 @@ module appc-provider-lcm {
1314 1315       }
1315 1316   }
1316 1317
     1318  +  /*****************************************************************************
     1319  +   * Define the traffic migration service
     1320  +   *****************************************************************************/
     1321  +      rpc migrate-traffic {
     1322  +          description "An operation to migrate traffic from a VM";
     1323  +          input {
     1324  +              uses common-header;
     1325  +              leaf action {
     1326  +                  type action;
     1327  +                  mandatory true;
     1328  +              }
     1329  +              uses action-identifiers;
     1330  +              leaf payload {
     1331  +                  type payload;
     1332  +                  mandatory true;
     1333  +              }
     1334  +          }
     1335  +          output {
     1336  +              uses common-header;
     1337  +              uses status;
     1338  +          }
     1339  +      }
```

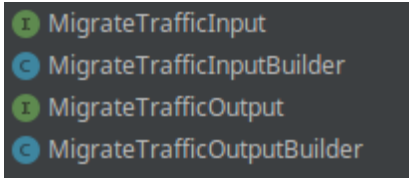
Compilation of YANG models in APPC

- You need to compile appc module now in order to generate Java classes based on YANG models
- Use: mvn clean install from appc/ directory
- Maven will generate following classes (--->)
- Note! The compilation will fail, but you will see new classes generated

appc/appc-provider/appc-provider-model/target/generated-sources/yang-gen-sal/org/opendaylight/yang/gen/v1/org/onap/appc/lcm/rev160108/AppcProviderLcmService.java

```
/**
 * An operation to migrate traffic from a VM
 *
 */
@CheckReturnValue
Future<RpcResult<MigrateTrafficOutput>> migrateTraffic(MigrateTrafficInput input);
```

appc/appc-provider/appc-provider-model/target/generated-sources/yang-gen-sal/org/opendaylight/yang/gen/v1/org/onap/appc/lcm/rev160108




```
public interface MigrateTrafficOutput
    extends
    CommonHeader,
    Status,
    DataObject,
    Augmentable<org.opendaylight.yang.gen.v1.org.onap.ccsdk.sli.northbound.lcm.rev180329.MigrateTrafficOutput>
```

```
public interface MigrateTrafficInput
    extends
    CommonHeader,
    ActionIdentifiers,
    DataObject,
    Augmentable<org.opendaylight.yang.gen.v1.org.onap.ccsdk.sli.northbound.lcm.rev180329.MigrateTrafficInput>
```

Implement new Service to handle RPC method

- Create new class in appc-provider module, example:

  appc/appc-provider/appc-provider-bundle/src/main/java/org/onap/appc/provider/lcm/service/MigrateTrafficService.java

- This class must:
  - extend AbstractBaseService
  - invoke superclass constructor with proper Action name (e.g. Action.MigrateTraffic)
  - handle the request
    - In this example this Service will process, validate and executeAction defined in DG
- Implementation example on the next slide.

```java
/**
 * Provide LCM command service for Migrate Traffic from VNF.
 */
public class MigrateTrafficService extends AbstractBaseService {

    /**
     * Constructor
     */
    public MigrateTrafficService() {
        super(Action.MigrateTraffic);
        logger.debug( msg: "MigrateTrafficService starts");
    }

    public MigrateTrafficOutputBuilder process(MigrateTrafficInput input) {
        // TODO: PoC only
        CommonHeader commonHeader = input.getCommonHeader();
        ActionIdentifiers actionIdentifiers = input.getActionIdentifiers();
        Payload payload = input.getPayload();

        validate(commonHeader, input.getAction(), actionIdentifiers, payload);

        if (status == null) {
            proceedAction(commonHeader,actionIdentifiers,payload);
        }

        MigrateTrafficOutputBuilder outputBuilder = new MigrateTrafficOutputBuilder(input);
        outputBuilder.setStatus(status);
        outputBuilder.setCommonHeader(input.getCommonHeader());
        return outputBuilder;
    }
```

```java
void validate(CommonHeader commonHeader,
              Action action,
              ActionIdentifiers actionIdentifiers,
              Payload payload) {
    // FIXME: Mock for now.

}

void proceedAction(CommonHeader commonHeader,
                   ActionIdentifiers actionIdentifiers,
                   Payload payload) {
    // TODO: PoC only
    RequestHandlerInput requestHandlerInput = getRequestHandlerInput(commonHeader, actionIdentifiers, payload,
            this.getClass().getName());
    if (requestHandlerInput != null) {
        executeAction(requestHandlerInput);
    }
}
```

Implement new method in AppcProviderLcm.java

- Implement new method from LCMService.java interface – example below (migrateTraffic)
- You should use a newly created Service to handle this RPC method, example:
  - new MigrateTrafficService().process(input);
- Do not forget to return Future result

appc/appc-provider/appc-provider-bundle/src/main/java/org/onap/appc/provider/AppcProviderLcm.java

```
@Override
public Future<RpcResult<MigrateTrafficOutput>> migrateTraffic(MigrateTrafficInput input) {
    logger.debug(String.format("LCM MigrateTraffic, received input: %s", input.toString()));
    MigrateTrafficOutputBuilder outputBuilder = new MigrateTrafficService().process(input);
    RpcResult<MigrateTrafficOutput> result =
            RpcResultBuilder.<~>status(true).withResult(outputBuilder.build()).build();
    return Futures.immediateFuture(result);
}
```

Add new method to VnfOperation list

- You need to append new action in one more place.. in order to allow APPC to recognize proper VNF Operation when invoking an API

appc/appc-dispatcher/appc-dispatcher-common/domain-model-lib/src/main/java/org/onap/appc/domainmodel/lcm/VNFOperation.java

appc-dispatcher/appc-dispatcher-common/domain-model-lib/src/main/java/org/onap/appc/doma...

```
...  ...   @@ -45,6 +45,7 @@ public enum VNFOperation {
45   45        Query,
46   46        QuiesceTraffic,
47   47        ResumeTraffic,
     48  +     MigrateTraffic,
48   49        Reboot,
49   50        Rebuild,
50   51        Restart,
```

Implement Directed Graph with service logic and configure APPC to activate it at bootup time

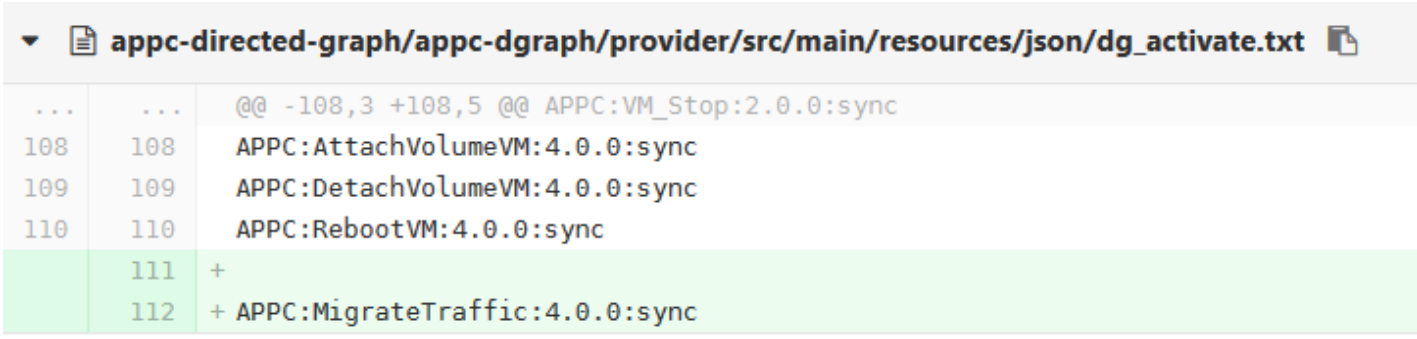
- You need to generate Directed Graph in the same way as for SDN-C (look at slide #11)
- Download JSON file and copy it to:

appc/appc-directed-graph/appc-dgraph/provider/src/main/resources/json

- Download XML file and copy it to:

appc/appc-directed-graph/appc-dgraph/provider/src/main/resources/xml

- Change configuration as in the picture below:
  - **Module name: APPC**
  - **Method name: MigrateTraffic**
  - **Version: 4.0.0**



Installation and deployment

- Compile APPC module
- Deploy APPC Docker containers

  (from appc/deployment/)

  mvn clean install -P docker

  (you should see new images in local Docker env)

  (from appc/docker-compose)

  docker-compose up –d

  (wait 5-10 min for APPC to start, check logs)

  docker-compose logs –f
- Check if new API exists in APIDOC explorer (passwd: admin/admin):

http://localhost:8282/apidoc/explorer/index.html