OpenECOMP Application Controller

User Guide

Revision **1.0.0**
Revision Date **17 January 2017**

Date	Revision	Author	Changes
2017-01-17	R1610.6	mjf	“OpenECOMP”, cosmetic changes
2017-02-06	R1610.7	mjf	copyright updated

Table of Contents

0. Preface

This preface contains:

0.1 Target Audience

This document is intended for an advanced technical audience, which includes engineers and technicians. This document will be revised when new versions of the software are released.

0.2 Accessing Commands and Domain Name System

Commands and Domain Name System (DNS) are provided in this document.

- If you are viewing the document online, click the example name.
- If you are viewing a printed copy of this document, look up the referenced example in the appropriate Appendix.

0.3 Related Documentation

The following sections describe the conventions this document uses, including notices, text conventions, and command-line conventions.

0.3.1 Notice

Note: Notes provide information of special interest or recommendations.

0.3.2 Command-line Conventions

You may encounter one or more of the following elements in a command-line path.

Table 0-1 Command-line Conventions

Convention	Description
Brackets [ ]	This is used for optional items.
Braces { }	This indicates choices separated by pipe (|) for sets from which only one is selected. For example: {even|odd}
Blue text	This indicates a link (that is, if you are viewing this document online).

0.3.3 Text Conventions

You may encounter the following text conventions in this document.

Table 0-2 Text Conventions

Convention	Description

Monospace font with blue shading	This font indicates sample codes, screenshots, or elements. For example:
	`contact": {`
	`"contactType": "USER",`
	`"source": "app1",`
	`}`

Italics	• Emphasizes a point or denotes new terms at the place where they are defined in the text. • Indicates an external book title reference.
Numeric	A number made up of digits 0 through 9.
Text	Any combination of alphanumeric characters. New items in RED

0.4 Contact Information

Refer to the Mailing Lists page on the OpenECOMP developer wiki for the Application Controller working group email address.

0.5 Authors and Contributors

The following table consists of the persons who are authors and contributors to this document.

0.5.1 Table 0-3 Contributors

Contributors	

0.6 Terms and Acronyms

The following table defines terms and acronyms used in this document.

1. Application Controller

This section discusses the Application Controller (APPC) application implementation and requirements.

1.1 Overview

The Application Controller (APPC) is one of the components of the Open Enhanced Control, Orchestration, Management, and Policy (OpenECOMP) platform, and is responsible for handling the Life Cycle Management (LCM) of Virtual Network Functions (VNFs). This document provides give detailed guidance on how to use APPC for LCM actions.

1.2 Implementation

The APPC infrastructure is implemented on virtual machines in an open stack cloud.

1.3 Features

The APPC HTTP API supports Life Cycle Management commands, allowing users to manage virtual applications and their components.

1.4 Application Controller Interface

1.4.1 Overview

The Application Controller Dashboard interacts with the controller using REST APIs and performs actions on VF/VMs, such as snapshot, lock, sync, and health-check.

The Application Controller Karaf Web Console interacts with the controller and provides a graphical overview of the runtime.
You can use it to:

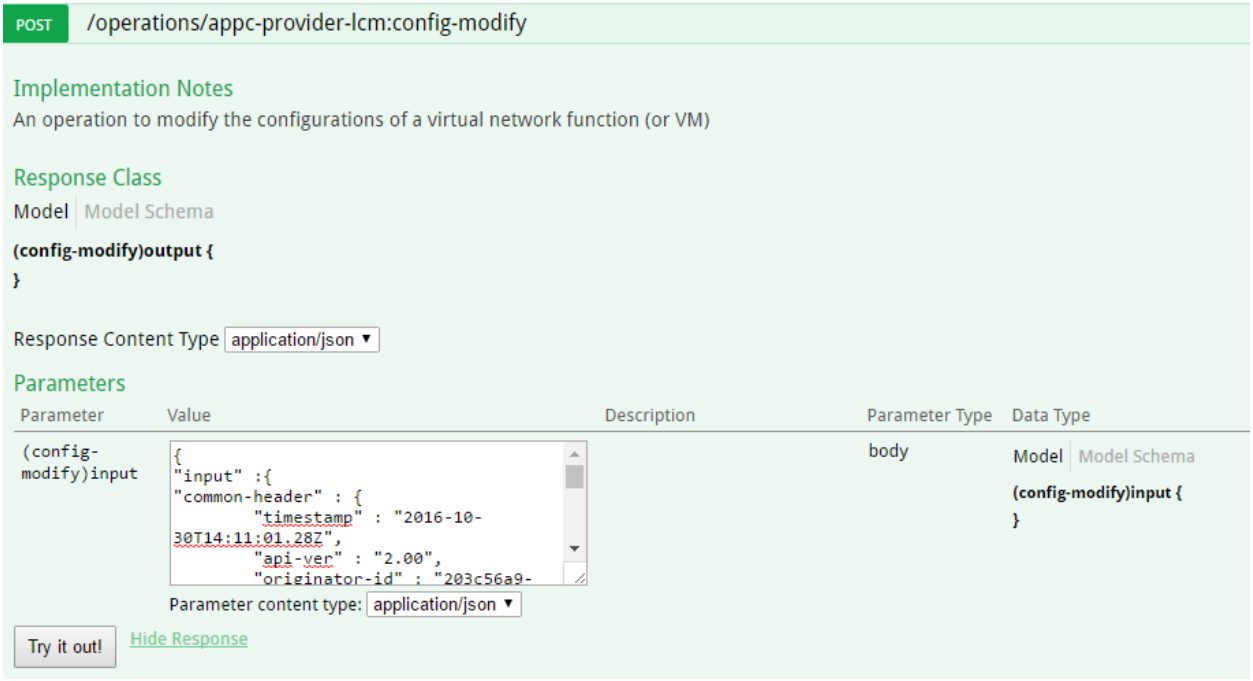
- Install and uninstall features
- Start, stop, install bundles
- Configure Karaf

1.4.2 Dashboard

To open the Application Controller dashboard, go to:

https://<controller-ip>:8443/apidoc/explorer/index.html

Navigate to the available LCM commands by clicking on **appc-provider-lcm**:



Click on the URI of the desired action to open a frame with information about the action and an option to try it out. For example, to send the action, add the request body as the `input` parameter value, and click **Try it out!**

The following figure shows an example body of a `config-modify` request:

```
{
 "input" :{
    "common-header" : {
        "timestamp" : "2016-08-30T10:02:08.97Z",
        "api-ver" : "1",
        "originator-id" : "c09ac7d1-de62-0016-2000-e63702155555",
        "request-id" : "c09ac7d1-de62-0016-1619-e637021cfc9",
        "sub-request-id" : "100",
        "flags" : {
            "force":"TRUE",
            "ttl":"65000"
        }
    },
    "action":"ConfigModify",
    "action-identifiers" : {
       "vnf-id" : "mock_vf"
    },
    "payload" : "{\"configuration-file-name\": \"MockModifyConfig\",
        \"vnf-type\": \"mock\",
        \"vnf-host-ip-address\": \"192.168.1.1\" }"
 }
}
```

If the request is accepted, you should see the following response:

Request URL

```
https://10.147.48.16:8443/restconf/operations/appc-provider-lcm:config-modify
```

Response Body


```
{
  "output": {
    "status": {
      "message": "ACCEPTED - request accepted",
      "code": 100
    },
    "common-header": {
      "flags": {
        "force": "TRUE",
        "ttl": 65000
      },
      "sub-request-id": "0dc24c2f-34e5-4177-9d15-18cae7c86188",
      "request-id": "fbe6fa1c-5457-45b8-b569-ca36d41fd654",
      "api-ver": "2.00",
      "originator-id": "203c56a9-61e1-44f0-8a02-df8e3fe43ae7",
      "timestamp": "2016-10-30T14:11:01.28Z"
    }
  }
}
```


Response Code

```
200
```

Response Headers


```
{
  "Access-Control-Allow-Origin": "https://10.147.48.16:8443",
  "Access-Control-Expose-Headers": "location",
  "Access-Control-Allow-Credentials": "true",
  "Server": "Jetty(8.1.18.v20150929)",
  "Transfer-Encoding": "chunked",
  "Content-Type": "application/json"
}
```

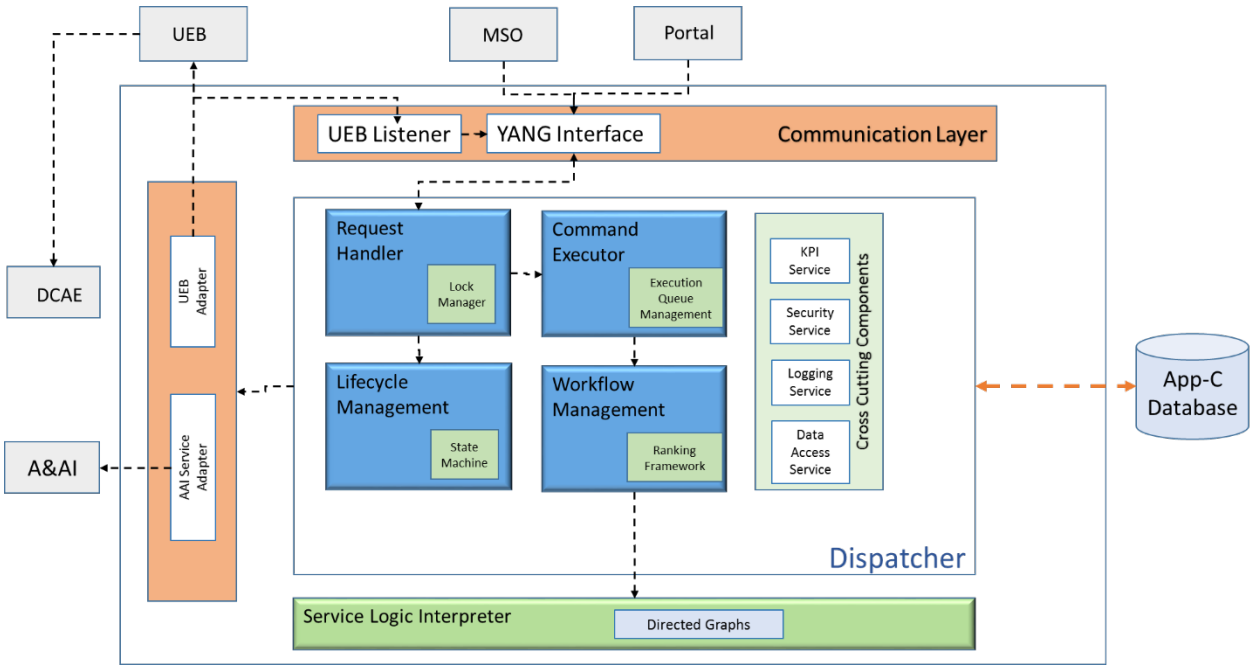
1.4.3 Karaf Web Console

To access the console for an instance of Application Controller, enter the following address in your web browser: https://<controller-ip>:8443/system/console
Log in with the username `karaf` and the password `karaf`. If you have changed the default user or password, use the one you have configured.

1.5 Architecture

This section discusses the APPC internal components in detail.

APP-C High Level Architecture



1.5.1 Dispatcher

The APPC Dispatcher component processes requests received by the LCM API REST handler from other OpenECOMP components. The Dispatcher checks the conditions are sufficient for performing the request and selects the correct Direct Graph (DG) workflow for execution, or rejects the request. When the DG execution is complete, the Dispatching function is responsible

for notifying the initiator of the operation with the request execution result (Success/Error) and updates the VNF state in Active and Available Inventory (A&AI).

The detailed responsibilities of the Dispatcher are described as follows:

- Upon receiving the operation request, the Dispatcher performs the synchronous part of the execution:
    - Starts the operation's time-to-live countdown
    - Queries A&AI to get the VNF type and its current operational state
    - Operates an LCM State Machine, which uses VNF_type specific rules to allow or reject the requested command execution
    - Allocates and initiates an appropriate Directed Graph (DG) workflow to start the asynchronous part of the execution
    - Returns a response to the initiator: OK or reject (for example, the State Machine blocks the operation, no DG or time-to-live, or bad parameters).
    - If the operation is rejected, the Dispatcher generates an appropriate Audit log for the Event and Error Logging Framework (EELF) and the Local Event Journal
- Upon workflow completion, the Dispatcher:
    - Receives the execution results from the DG
    - Publishes the execution result over UEB (Success or error)
    - Updates VNF status in A&AI
    - Generates an Audit log for EELF and Local Event Journal

1.5.2 State Machine

The VNF State machine enables the Dispatching function to determine the validity of the requested operation (desired state) as a function of the current VNF state, acquired from the A&AI. The State machine maintains its data (states and valid operations) in-memory. At the point of APP-C initialization, the State Machine constructs a matrix based on the metadata of the current operation and the valid desired state.

1.5.3 APPC Provider

The APPC Provider module exposes the endpoints for each action supported by APPC. This module uses the YANG model to define the YANG Remote Processing Call (RPC) and data model, in other words, the input and output parameters for each action. The Provider module is responsible for validating the RPC input and for rejecting any malformed input. After successful validation, the APPC Provider calls the Dispatcher to continue the request processing.

1.5.4 The Service Logic Interpreter (SLI) Framework

The SLI framework is responsible for executing Directed Graphs (DGs). The Dispatcher invokes the SLI framework to execute a specific DG based on the input action. The SLI executes the DG execution and the sub-DG returns a success or failure response to the caller along with context variables used to exchange data with the calling body (for example, the Dispatcher). The caller can use the SLI context to specify data required for DG execution. The same context is returned from the DG once execution is complete.

1.5.5 A&AI

A&AI is used as a source for the VNF status and topology. It is also used to update the VNF status at the end of the operation: for example, VNFC record(s) are added after configuration. The A&AI status is not updated for read-only operations, such as Sync or Audit. In the case of the Terminate operation, APP-C also removes the terminated VNF from A&AI by deleting its Virtual Machines (VM)s.

Access to the A&AI is conducted using the SDN-C A&AI adapter via the Dispatching function and operation-specific DGs.

1.5.6 Southbound VNF Adapters

APPC uses several adapters to connect to VNFs. The IAAS adapter is provided with the ODL platforms. Other adapters have been added.

1.5.6.1 IAAS Adapter

IAAS Adapter is the southbound adapter for APP-C. It connects with the OpenDaylight controller to perform various operations on VNFs such as restart, migrate, rebuild etc. The IAAS Adapter is effectively used as a DG plugin in that the services exposed by the adapter are called from DGs.

1.5.7 SSH (XML/CLI) Adapter

A custom adapter has been added which can connect to a VNF using an SSH session. It is designed to support CLI and XML protocols, including Netconf. It is used to load configurations and retrieve the running configuration.

1.5.8 Transactions store

For each operation request procedure that completes or terminates, APPC generates and stores an accurate transaction record in its internal database, including:

- Timestamp
- Request ID
- Start time
- End time
- VF_ID
- VF_type
- Sub-component (optional) e.g. VFC_ID/VM UUID
- Operation: for example Start, Configure, etc.
- Result: Success/Error code and description, as published to the initiator

1.5.9 Table 1-1 Application Controller Interface Table

Source	Flow	Destination	Service	Port	Purpose / Comments	Frequency
APPC	→	A&AI	REST	8443	APPC retrieves and updates the VNF data in AAI.	As needed
APPC	→	SLI	Java (internal)	N/A	APPC sends the LCM API request to SLI for DG execution	As needed
APPC	→	South-bound Adapters	Java (internal)	N/A	APPC interacts with southbound adapters for VNF Lifecycle Management Actions	As needed
APPC	→	UEB	JMS	3904	APPC sends the Asynchronous responses and Failure events to UEB	As needed

2. Application Controller Releases

The Application Controller is released for each OpenECOMP Release

2.1 Load and Performance Testing

Load and performance testing is done prior to E2E/Production deployment.

3. Logging

3.1 Logging in the APPC Application

APP-C uses Event and Error Logging Framework (EELF) for application logs.

To enable EELF logging:

1. Replace the default configuration file located at
   `/opt/opendaylight/current/etc/org.ops4j.pax.logging.cfg`
   with the configuration file that is checked into git
2. Stop and restart `ODL controller` for the configuration changes to take effect.
3. Verify logging changes at the following log paths:
   - /`opt/opendaylight/current/data/log/eelf/karaf.log`
     This log contains the regular karaf.log output reformatted to use the EELF MDC properties and the pattern that is configured in the `org.ops4j.pax.logging.cfg` file.
   - `/opt/opendaylight/current/data/log/APPC/`**`<package-name>`**
     This directory contains the audit, metric, error, and debug logs that are configured in the `org.ops4j.pax.logging.cfg` file.

     Note: `/opt/opendaylight/current/data/log/APPC/`**`controller`** contains the logs generated from the package `org.openecomp.*` (all APPC logs)

- Error.log: alarms –ERROR level logs and above
- Info.log: INFO level logs only
- Debug.log: debugging – DEBUG level and above
- Audit – AUDIT level and above

4. Monitoring

The APPC does not currently support monitoring by the Nagios monitoring software.

5. APPC Troubleshooting Tips and Production M&Ps

5.1 Troubleshooting Tips

TBD

5.2 Production M&Ps

TBD

5.3 Deployment Process

The APPC Deployment Guide covers the deployment process.

6. APPC Use and Configuration

6.1 LCM Action Request and Response

The APPC exposes an HTTP API to support the Life Cycle Management (LCM) commands sent from OpenECOMP components such as MSO, DCAE, and the Portal. These commands enable the components to request APPC to perform actions such as to control, modify, start, or stop virtual applications and/or their components.

A virtual application is composed of a maximum of four layers:

- Service
- Virtual Network Function (VNF)
- Virtual Network Function Component (VNFC)
- Virtual Machine (VM)

A Life Cycle Management command may affect any number of these layers.

APPC supports the following types of LCM requests:

6.1.1 LCM over REST (HTTP POST)

LCM command requests over REST are sent to the APPC using an HTTP POST request. The APPC returns one or more responses for each LCM request.

An **asynchronous** command, containing an authorized and valid request, results in at least two discrete response events:

- an ACCEPT response to indicate that the request is accepted and will be processed, and
- a final response for the command containing a final status.

An unauthorized or invalid request would result in a single ERROR response.

A **synchronous** command, for example Lock or Unlock, results in a single response, which is either SUCCESS or ERROR. For this type of request, the first response is a synchronous HTTP response. The second final status response is sent over the UEB.

The APPC API provides a POST HTTP API endpoint per command.

For the 1610 release, APPC supports the https protocol, whereas plain http requests are blocked.

Endpoint format:

```
<http-protocol>://<appc-ip>:<appc-api-port>/restconf/operations/appc-provider-lcm:<command-name>
```

To see an example, go to:
https://10.147.113.151:8443/restconf/operations/appc-provider:config-modify

6.1.2 LCM over UEB (the OpenECOMP bus)

LCM command requests over UEB are sent as messages on the OpenECOMP bus (UEB/DMaaP). APPC returns one or more responses for each LCM request.

An **asynchronous** command containing an authorized and valid request, results in at least two discrete response events:

- an ACCEPT response to indicate that the request is accepted and will be processed, and
- a final response for the command containing a final status.

An unauthorized or invalid request would result in a single ERROR response.

A **synchronous** command, for example Lock or Unlock, results in a single response, which is either SUCCESS or ERROR. For this type of request, both responses are returned via UEB/DMaaP. The Read / Write topics for the UEB need to be configured as described in Universal Event Bus (UEB).

For further information about the request and response format, see the APPC API Guide.

6.2 APPC Setup and Configuration

6.2.1 Universal Event Bus (UEB)

APPC sends asynchronous responses using the Universal Event Bus (UEB). It also receives requests from UEB (see the APPC API Guide for further details). The UEB Adapter Bundle handles all UEB operations (send / receive messages), and requires the following properties configured in `/opt/opendaylight/current/app/properties/appc.properties`:

```
# Asynchronous responses
ueb.topic.write=<WRITE_TOPIC> // e.g. async_demo
# e.g. 10.147.101.46:3904
ueb.poolMembers= <HOST_IP_1>:<PORT_NUMBER>,<HOST_IP_2>:<PORT_NUMBER>
#
# DG events (asynchronous) in case of failures
DCAE.event.topic.write=<WRITE_TOPIC> // e.g. event_demo
#
# e.g. 10.147.101.46:3904
DCAE.event.pool.members=<HOST_IP_1>:<PORT_NUMBER>,<HOST_IP_2>:<PORT_NUMBER>
#
# 1610 LCM API (rpc) - synchronous
# The following properties are required for sending LCM request over UEB.
appc.LCM.provider.url=https://localhost:8443/restconf/operations/appc-provider-lcm
# e.g. 10.147.101.46:3904
appc.LCM.poolMembers=<HOST_IP_1>:<PORT_NUMBER>,<HOST_IP_2>:<PORT_NUMBER>
#
appc.LCM.topic.read=<READ_TOPIC> // e.g. test2021
appc.LCM.topic.write=<WRITE_TOPIC> // e.g. APPC-TEST-LCM
appc.LCM.client.name=<CLIENT_NAME> // e.g name1
appc.LCM.client.name.id=<CLIENT_ID> // e.g 0
appc.LCM.provider.user=<LCM PROVIDER Username> // e.g. admin
appc.LCM.provider.pass=<LCM PROVIDER Username> // e.g. admin
```

6.2.2 A&AI

APPC connects with A&AI using the AAI service (`aai-service-karaf-extension-9.0.15.zip`). The current version of the AAI service library is 9.0.15.

To initialize AAI services, the following A&AI properties need to be configured in `/opt/opendaylight/current/app/properties/aaiclient.propertis`:

```
org.openecomp.sdnc.sli.aai.ssl.trust= <SSL KEY Store location> e.g.
//opt/opendaylight/current/app/tls-client/aai-client-truststore.jks
org.openecomp.sdnc.sli.aai.ssl.trust.psswd=<SSL KEY Store Password>
org.openecomp.sdnc.sli.aai.ssl.key= <SSL KEY location > e.g
//opt/opendaylight/current/app/tls-client/aai-client-cert.p12
org.openecomp.sdnc.sli.aai.ssl.key.psswd=<SSL KEY Password>
org.openecomp.sdnc.sli.aai.uri=https://<IP_ADDRESS>:<PORT_NUMBER>
```

6.2.3 Database connection

APPC uses dblib service (`dblib-karaf-extension-9.0.1.zip`) for all database operations. The current version of dblib service is 9.0.1. This library uses the file, `/opt/opendaylight/current/app/properties/dblib.properties`, which contains the requisite database properties, such as host, user and password:

```
org.openecomp.sdnc.sli.dbtype=jdbc
org.openecomp.sdnc.sli.jdbc.url=jdbc:mysql://<HOST_IP>:3306/<DB_NAME>
org.openecomp.sdnc.sli.jdbc.database=<DB_NAME>
org.openecomp.sdnc.sli.jdbc.user=<USER>
org.openecomp.sdnc.sli.jdbc.password=<PASSWORD>
org.openecomp.sdnc.sli.jdbc.limit.max=<CONNECTION POOL MAXIMUM SIZE> // e.g. 10
org.openecomp.sdnc.sli.jdbc.limit.min=<CONNECTION POOL MINIMUM SIZE> // e.g. 4
org.openecomp.sdnc.sli.jdbc.limit.init=<CONNECTION POOL INITIAL SIZE> // e.g. 5
org.openecomp.sdnc.sli.jdbc.connection.name=<CONNECTION_NAME>
org.openecomp.sdnc.sli.jdbc.hosts=<HOST>
```

6.2.4 APPC Transactions Database connection

```
org.openecomp.appc.db.url.appcctl=jdbc:mysql://<HOST_IP>:3306/appcctl
org.openecomp.appc.db.user.appcctl=appcctl
org.openecomp.appc.db.pass.appcctl=appcctl
```

6.2.5 SLI (SVC Logic)

APPC use the SLI service (`sli-karaf-extension-9.0.5.zip`) to execute the DG. The current version of SLI service is 9.0.5. To initialize SLI services, the following properties need to be configured in `/opt/opendaylight/current/app/properties/svclogic.properties`. The database operations performed from the DG also use this database configuration.

```
org.openecomp.sdnc.sli.dbtype = jdbc
org.openecomp.sdnc.sli.jdbc.url =jdbc:mysql://<HOST_IP>:3306/<DB_NAME>
// jdbc:mysql://localhost:3306/sdnctl
org.openecomp.sdnc.sli.jdbc.database =<DB_NAME> e.g. sdnctl
org.openecomp.sdnc.sli.jdbc.user = <USER> e.g. sdnctl
org.openecomp.sdnc.sli.jdbc.password = <PASSWORD>
```

6.2.6 IAAS Adapter

IAAS Adapter is the southbound adapter of APPC. To initialize the IAAS Adapter service, the following properties need to be configured

in `/opt/opendaylight/current/app/properties/appc.properties`:

```
# Provider (OpenStack) configuration
provider1.name=<Provider NAME>
provider1.identity=http://10.147.249.40:5000/v2.0
provider1.tenant1.name=<TENANT>
provider1.tenant1.userid=<USER_NAME>
provider1.tenant1.password=<PASSWORD>
```

7. Application Controller VNF Onboarding

7.1 LCM Command Execution Overview

The Application Controller assumes that the A&AI instance it is configured with contains all the information it needs about VNF/VNFC/VMs, otherwise any request by the user to perform an action on a VNF will fail. The Application Controller uses a variety of SQL tables in order to perform actions on a VNF, all of which are described in Creation of DGs

DGs are created using the Direct Graph Builder - Node Red graphical utility for DGs creation. DGs are then stored as XML files and loaded to APPC MySQL database. The DGs invoke the execution of Java code from different nodes.

DGs are resolved according to LCM Action, API version, VNF Type, and VNF Version.

The SLI framework is responsible for executing the DGs.

Data Setup.

Initially, Application Controller should have a set of DGs designed for the specific VNF type. These DGs are stored in the SVC_LOGIC table.

After a user sends an action request to the controller, the Application Controller uses the VNF_DG_MAPPING table to map the requested action to a specific DG. If the mapping was successful, the input body is validated and the user receives a synchronous response containing an Accept or a Reject message to indicate whether the request was rejected or whether it was accepted and the controller initiated the DG flow.

During the execution of a DG, the controller may use one or more SQL tables to fetch or store data. For example, in order to perform a ConfigModify action, the controller needs to fetch a username and password to connect to the VNF and change its configuration.

ALL tables used during DG execution are described below.

7.2 Creation of DGs

DGs are created using the Direct Graph Builder - Node Red graphical utility for DGs creation. DGs are then stored as XML files and loaded to APPC MySQL database. The DGs invoke the execution of Java code from different nodes.

DGs are resolved according to LCM Action, API version, VNF Type, and VNF Version.

The SLI framework is responsible for executing the DGs.

7.3 Data Setup

APP-C uses MySQL database as a persistent store. This section describes the tables in general and the tables that require data to be set up before sending a request.

7.3.1 SVC_LOGIC

This table stores all NodeRed DGs invoked by actions executed by APPC. The SLI framework uses this table for running the DG. If the DG does not exist in this table, the SLI framework returns a 'DG not found' error.

7.3.1.1 Table 7-1: Parameters

module	rpc	version	mode	active	graph
APPC	Generic_Audit	2.0.0	sync	N	<BLOB>

`module, rpc, version`

The `module, rpc,` and `version` parameters uniquely identify a Directed Graph (DG). The SLI framework uses these three parameters to invoke a DG or sub-DG. By convention, for the main DG, `rpc` is a combination of 'VNF type' (the generic type applied to all VNFs) followed by '_' and 'action'. This is the default convention; resolution of the DG for specific actions is handled individually in the relevant forthcoming sections.

`mode`

The DG execution node. This value is set to ‘sync’ for all APPC DGs.

`active`

This flag is the value of either 'Y' or 'N'. This flag is only used if specific version of DG is not mentioned while calling DG. If version of DG is not mentioned SLI framework will look for DG with active Flag set to 'Y' and execute it if found.

`graph`

This is actual graph invoked by SLI framework. The data type is SQL BLOB.

7.3.1.2 Loading Data into SVC_LOGIC

Follow these steps to load data in the table:

1. Copy the following libraries to a folder such as `C:\Domain2\lib` (these libraries can be located in nexus repository):

```
antlr4-4.5.1.jar
antlr4-runtime-4.5.1.jar
appc-dg-provider-10.0.1.jar
appc-dg-swm-10.0.1.jar
commons-io-2.5.jar
commons-lang-2.6.jar
commons-lang3-3.1.jar
dblib.properties
dblib-provider-9.0.1.jar
guava-18.0.jar
java-concurrent-hash-trie-map-0.2.23.jar
jcl-over-slf4j-1.6.1.jar
jsr305-3.0.0.jar
load_dg.sh
mysql-connector-java-5.1.39.jar
slf4j-api-1.6.1.jar
slf4j-simple-1.7.5.jar
sli-common-9.0.5.jar
sli-provider-9.0.5.jar
```

2. Place all XML files that need to be loaded in any folder, for example `XML\`.
3. Run the following command to load DGs:
   `java -classpath lib\* org.openecomp.appc.dg.DGXMLLoadNActivate XML\ activate.txt dblib.properties`
   Modify the three parameters as required:
   - `XML\`: The folder where all DG XMLs are placed in step 2
   - `activate.txt`: The file that lists the DGs that need to be activated. For example, in the SVC_LOGIC table, mark ACTIVE = 'Y'. This is optional.
     The format is: `module:rpc:version:mode`.
   - `dblib.properties`: The properties file with database details. Ensure that you update the database IP in this file to your APPC instance (where you need to load DGs).

   Parameters in dblib.properties file:

```
org.openecomp.sdnc.sli.dbtype = jdbc
org.openecomp.sdnc.sli.jdbc.url = jdbc:mysql://<HOST-IP>:3306/sdnctl
org.openecomp.sdnc.sli.jdbc.database = sdnctl
org.openecomp.sdnc.sli.jdbc.user = sdnctl
org.openecomp.sdnc.sli.jdbc.password = gamma
org.openecomp.sdnc.sli.jdbc.limit.max=10
org.openecomp.sdnc.sli.jdbc.limit.min=4
org.openecomp.sdnc.sli.jdbc.limit.init=5
org.openecomp.sdnc.sli.jdbc.connection.name=sdnctl
org.openecomp.sdnc.sli.jdbc.hosts=<HOST-IP>
```

4. The expected output for successful load

```
******************** Loading DG into Database ****************************
Loading DG XML file :C:\Domain2\XML\APPC_2.0.0_method_Generic_Audit.xml
[main] INFO org.openecomp.sdnc.sli.SvcLogicParser - Saving SvcLogicGraph to database
(module:APPC,rpc:Generic_Audit,version:2.0.0,mode:sync)
******************** Activating DG into Database ****************************
Activating DG :APPC:Generic_Audit:2.0.0:sync
Found Graph :APPC:Generic_Audit:2.0.0:sync Activating ...
```

Note: Any validation errors will be displayed in the console.

7.3.2 VNF_DG_MAPPING

This table stores the VNF and its corresponding DG. This is used by the DG resolver logic of the Dispatcher to map the DG to the requested action. Only the mapping is stored; the actual DG is stored in the SVC_LOGIC table.

The DG resolver logic uses a combination of `action`, `api_version` and `vnf_type` to retrieve the DG details: `dg_name` (rpc column of SVC_LOGIC table), `dg_version` and `dg_module`.
The `module`, `rpc` and `version` uniquely identify the DG.

Blank, null or '*' values in `api_version` , `vnf_type` and `vnf_version` are matched with any values by the DG resolver. For example, a generic DG which can be invoked on any type of VNF 'vnf_type' can be blank / null or *. The DG resolver logic first tries to match a specific DG, and if this is not found, then look for a generic match using '*'. For example, an entry for the Test action and vnf_type VSBG is specific, so it is only used for VNFs of type VSBG, whereas for the Sync action the same DG is used for any type of VNF and any version.

7.3.2.1 Table 7-2: Parameters

action	api_version	vnf_type	vnf_version	dg_name	dg_version	dg_module
Test	2	VSBG		VSBG_Test	2.0.0.1	APPC
Sync				Generic_Sync	2.0.0	APPC

7.3.3 DEVICE_AUTHENTICATION

This table stores device authentication details. It is used by actions such as Audit and Sync which connect with VNFs. This table stores a record that corresponds to each VNF type, so `vnf_type` is unique.

Username, password and port_number are fields corresponding to `vnf_type`.

7.3.3.1 Table 7-3: Parameters

DEVICE_AUTHENTICATION_ID	VNF_TYPE	USER_NAME	PASSWORD	PORT_NUMBER
41	vDBE-V	root	<password>	22

7.3.4 VNF_LOCK_MANAGEMENT

This table is used to persist data for vnf locking. APP-C lock vnf id when action start executing on that vnf id. This table stores vnf_id i.e. resource_id along with owner , expiration_time or timeout. before execution of request dispatcher check if VNF_ID is already locked by another action in execution. If not locked dispatcher will lock vnf else return VNF locked error to caller.

7.3.4.1 Table 7-4: Parameters

RESOURCE_ID	OWNER_ID	UPDATED	TIMEOUT	VER
AUDIT_1652	vDBE-V	1474457140000	0	22

This table do not require any initial setup.

7.3.5 VNF_STATE_MANAGEMENT

This table is used to store the operational state of VNF_ID, whether it is stable or unstable. It stores `state`, `owner` and `updated` time (in milliseconds). This table does not require any initial setup.

7.3.5.1 Table 7-5: Parameters

VNF_IF	STATE	OWNER_ID	UPDATED	TIMEOUT	VER
ASHISH_VSBG_ VNFS_1787	STABLE	ORIG_1787@REQ_1787 @SUBREQ_1787	1474457140000	0	22

7.3.6 UPLOAD_CONFIG

This table is used by configuration management actions such as Audit, Sync, ConfigModify, Terminate, and similar. It stores device configuration: one row or record corresponds to one VNFC, so therefore a VNF that has multiple VNFCs has multiple rows in the table.
The UPLOAD_CONFIG table stores configuration as the following types:

- `Current`
- `Running`

- `Historic`

The `config_indicator` column denotes the type of configuration, where a null value denotes `Historic` configuration. For a single VNFC there should only be one `Current` and one `Running` configuration , but there can be multiple `Historic` configurations. This table does not require any initial setup.

7.3.6.1 Table 7-6: Parameters

Parameter Name	Value for ConfigModify	Value for Sync
UPLOAD_CONFIG_ID	63	67
REQUEST_ID	3	REQ_1690
ORIGINATOR_ID	12345	ORIG_1690
SERVICE_DESCRIPTION	abcde	abcde
ACTION	ConfigModify	Sync
UPLOAD_DATE	2016-08-01 14:30:40	2016-09-22 12:30:40
VNF_ID	AUDIT_1767	AUDIT_1690
VNF_NAME	GET_RUN_CONFIG_VNF	GET_RUN_CONFIG_VNF
VM_NAME	GET_RUN_CONFIG_VNF	GET_RUN_CONFIG_VNF
VNF_TYPE	vDBE-V	vDBE-V
VNFC_TYPE	vDBE-V2	vDBE-V1
HOST_IP_ADDRESS	135.25.69.37	
CONFIG_INDICATOR	Current	Running
PENDING_DELETE		
CONTENT	Dummy_current	<Configuration>

7.3.7 DEVICE_INTERFACE_PROTOCOL

This table is used by the 'getRunningConfig' DG, which is a sub-DG, called by Audit and Sync DG. It stores the VNF type and corresponding sub-DG, which are used to get the running configuration of a device. The `module` and `DG_RPC` are used to identify the DG from the SVC_LOGIC table. The `protocol` is used to store the protocol defined for retrieving configuration. If a mapping between the VNF type and the DG does not exist in this table, then actions such as Audit and Sync fail with the error message 'Device Interfacing DG not available'.

7.3.7.1 Table 7-7: Parameters

DEVICE_INTERFACE_PROTOCOL_ID	VNF_TYPE	PROTOCOL	MODULE	DG_RPC
4	vDBE-V	NETCONF-XML	APPC	getDeviceRunningConfig

7.3.8 CONFIGFILES

This table is used by the several configuration DGs, using a legacy configuration API, to store artifacts from SDC, configuration data from requests, and configurations to be downloaded to VNFs.

7.3.8.1 Table 7-8: Parameters

Parameter Name	Value
CONFIG_FILE_ID	24
EXTERNAL_VERSION	
DATA_SOURCE	Configurator
CREATION_DATE	6/9/2016 11:16:57 AM
SERVICE_INSTANCE_ID	ibcx0001vm001
VNF_TYPE	ISBC
VNFC_TYPE	vISBC - mmc

Parameter Name	Value
FILE_CATEGORY	device_configuration
FILE_NAME	orch_config.json
FILE_CONTENT	(contains configuration)

7.3.9 GET_CONFIG_TEMPLATE

This table is used by the 'getDeviceRunningConfig' DG which is used as a sub-DG for the Audit and Sync actions. It stores template data corresponding to the VNF type. Template data is only used when the protocol in DEVICE_INTERFACING_PROTOCOL table is set to 'CLI'. Other protocols do not use this table. If Data does not exist in this table and protocol is set to 'CLI' then DG results in 'Error getting Template Data'.

7.3.9.1 Table 7-9: Parameters

Parameter Name	Value
GET_CONFIG_TEMPLATE_ID	1
VNF_TYPE	Generic
DEVICE_INTERFACE_PROTOCOL_ID	3
XML_PROCESSING	
XML_PROTOCOL	
TEMPLATE	Login_Prompt: Matches "Login as:"...

8. VNF Configuration Management

APPC supports LCM actions for configuration management, such as Sync, Audit, and ConfigModify. APPC persists device configuration to the APPC MySQL database. All configuration management actions manage this data (read / write / update) during their operations.