Micro-services friendly nimble distribution and extensions

ODL Magnesium DDF, Antwerp - September 2019

Luis Gomez & Tejas Nevrekar, Lumina Networks
Agenda

• Background - The Why
• Our approach
• Progress
• Goals for Magnesium
• Execution Plan
Background - Microservices

- Application definitions becoming static, less use of adding/removing application functions at runtime
- New “services” come up in new containers and advertise to a service registry
- Consumers can use the registry to interact with such new services
- Old services just retired by stopping the containers
- Given that containers mostly run on clouds that spend $$$ per usage, need optimal images, every second counts (Ref: Appendix)
Background - ODL History

• As ODL starts getting into microservice, containerized deployments, the ask to make ODL nimble
  • Reduce memory, CPU footprint
  • Reduce startup time
  • Optimize key code paths

• Alternatives to solve this problem:
  • ODL simple - https://github.com/vorburger/opendaylight-simple
    • Michael already did upstream some work and the rest can be upstreamed any time.
  • Lighty.io - https://github.com/PantheonTechnologies/lighty-core
    • Solves these issues already, but not upstreamed in ODL and no public plan to do this.
Our Approach

• Our controller distribution is very close to the upstream distribution
• As the scale requirements are increasing, we feel the need to have such non-karaf distribution in ODL upstream
• It is not possible for us to use a 3rd party github project as the basis of our ODL distribution
• We need the changes for distribution to be in ODL upstream
• That leaves ODL simple as the only option for us to align with
Progress made by odl-simple upto Neon

- Upstream changes:
  - Infrautils - annotation processing to enable guice wiring
  - Moved from blueprint to javax.annotations - openflowplugin, ovsdb, genius

- github repo - https://github.com/vorburger/opendaylight-simple contains Wiring & Module classes
  - aaa, controller, genius, infrautils, mdsal, netvirt, openflowplugin, ovsdb, restconf, serviceutils
  - Wiring - Does the stitching of required external configuration as present in the blueprint.xml
  - Module - Guice implementation to expose the required services
Goals (for Magnesium Release)

• Expand the scope of odl-simple to cover “Managed” and stable “Self-Managed” projects, for example:
  • Netconf, BGPCEP, LISP, etc
  • JSON-RPC, TransportPCE

• Build smaller micro-distributions that contain smaller sets of modules suitable for micro-service deployments:
  • openflow simple: mdsal, controller, restconf, openflowplugin
  • netconf simple: mdsal, controller, restconf, netconf
  • bgpcep simple: mdsal, controller, restconf, bgpcep
  • netvirt simple: mdsal, controller, restconf, openflowplugin, ovsdb, genius, netvirt
Work - Development, Validation

- Add code in “Managed” and stable “Self-Managed” projects
  - This code has no impact in the project or the current Karaf/OSGI distribution
  - We will use Unit Tests to validate this code
- Add code for generating the micro-distributions
  - This code can be in the project repos or centralized
  - We can use exiting System Test (CSIT) to validate the new micro-distribution, a weekly distribution test would be enough
- Perform Benchmarking tests to compare with existing Karaf/OSGI distribution
  - Startup time
  - CPU
  - Memory footprint
- Revisit Backlog -
  https://github.com/vorburger/opendaylight-simple/blob/master/TODO.md
Execution Plan

• Move odl-simple github code in OpenDaylight
  • Michael Vorburger is on board with this
  • New Self-Managed project in Magnesium

• Make infra changes to deploy and test micro-distributions
  • Modify existing CSIT jobs to deploy micro-distributions should be straight forward

• Start generating & testing micro-distributions
  • As mentioned this will require few patches in existing ODL projects
Backup
## OLD Simple Runtime statistics

<table>
<thead>
<tr>
<th></th>
<th>Clean Start Time</th>
<th>Clean Init Time</th>
<th>Clean Time duration</th>
<th>Next Start Time</th>
<th>Next Init Time</th>
<th>Next Time Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODL-Neon-SR3</td>
<td>0:07:13</td>
<td>0:08:47</td>
<td>0:01:34</td>
<td>0:16:39</td>
<td>0:17:10</td>
<td>0:00:31</td>
</tr>
<tr>
<td></td>
<td>0:12:35</td>
<td>0:13:19</td>
<td>0:00:44</td>
<td>0:17:36</td>
<td>0:18:03</td>
<td>0:00:27</td>
</tr>
<tr>
<td></td>
<td>0:13:52</td>
<td>0:14:19</td>
<td>0:00:27</td>
<td>0:18:26</td>
<td>0:18:54</td>
<td>0:00:28</td>
</tr>
<tr>
<td></td>
<td>0:14:57</td>
<td>0:15:41</td>
<td>0:00:44</td>
<td>0:19:34</td>
<td>0:20:03</td>
<td>0:00:29</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>0:00:52</strong></td>
<td></td>
<td></td>
<td><strong>0:00:29</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Start Time</th>
<th>Init Time</th>
<th>Net Time Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODL-Simple-Neon-GA</td>
<td>1:44:16</td>
<td>1:44:28</td>
<td>0:00:12</td>
</tr>
<tr>
<td></td>
<td>1:46:02</td>
<td>1:46:15</td>
<td>0:00:13</td>
</tr>
<tr>
<td></td>
<td>1:46:35</td>
<td>1:46:46</td>
<td>0:00:11</td>
</tr>
<tr>
<td></td>
<td>1:47:29</td>
<td>1:47:40</td>
<td>0:00:11</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>0:00:12</strong></td>
</tr>
</tbody>
</table>
Key Implementation Steps

1. Replace all exposed blueprint XMLs with google.guice.AutowiringModule subclasses

2. Install required dependent modules

3. Expose required services using annotation for binding by odl:type

4. Refer required services using the annotation
Key Implementation Steps

5. Add AutoWiring class for reading configuration or initializing

```java
@Singleton
public class OpenFlowJavaWiring {

    public OpenFlowJavaWiring(ConfigReader configReader,
                                SwitchConnectionProviderFactory switchConnectionProviderFactory) {
        SwitchConnectionConfig defaultSwitchConnConfig = configReader.read("/initial/default-openflow-connection-config", SwitchConnectionConfig.class);
        SwitchConnectionProvider defaultSwitchConnProvider = switchConnectionFactory.newInstance(defaultSwitchConnConfig);
    }
```

6. Read from config using ConfigReader

```java
@Provides
@Singleton
SwitchConnectionProviderList getOpenFlowJavaWiring(OpenFlowJavaWiring openFlowJavaWiring) {
    return openFlowJavaWiring.getSwitchConnectionProviderList();
}
```

7. Use Wiring in the Module
Key Implementation Steps

8. Alternatively move most of the blueprint definition to Annotations in the Code in the respective project. E.g. as done in OVSDB - https://git.opendaylight.org/gerrit/c/ovsdb/+/79782

9. No additional explicit wiring needed in simple if blueprint.xml replaced by annotations.