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

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

	Start Time	Init Time	Net Time Duration
ODl-Simple-Neon-GA	1:44:16	1:44:28	0:00:12
	1:46:02	1:46:15	0:00:13
	1:46:35	1:46:46	0:00:11
	1:47:29	1:47:40	0:00:11
Average			**0:00:12**

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

```
public class NetconfModule extends AutoWiringModule {


    @Override
    protected void configureMore() {
        LOG.info("Loading netconf");
        // Guice
        install(new AnnotationsModule());
        // Controller/MD-SAL
        install(new InMemoryControllerModule());
```

```
@Provides
@Singleton
@GlobalWorkerGroup
EventLoopGroup getGlobalWorkerGroup() {
    return NioEventLoopGroupCloseable.newInstance(0);
}
```

```
@Provides
@Singleton
@org.opendaylight.netconf.simple.NetconfClientDispatcher
NetconfClientDispatcher getNetconfClientDispatcher(
        @GlobalBossGroup EventLoopGroup globalBossGroup,
        @GlobalWorkerGroup EventLoopGroup globalWorkerGroup,
        @GlobalTimer Timer globalTimer) {
    return new NetconfClientDispatcherImpl(globalBossGroup, g
}
```

Key Implementation Steps

5. Add AutoWiring class for reading configuration or initializing

```
@Singleton
public class OpenFlowJavaWiring {
```

6. Read from config using ConfigReader

```
@Inject
public OpenFlowJavaWiring(ConfigReader configReader,
        SwitchConnectionProviderFactory switchConnectionProviderFactory)
    SwitchConnectionConfig defaultSwitchConnConfig = configReader
            .read("/initial/default-openflow-connection-config", SwitchC
                    "openflow-switch-connection-provider-default-impl");
    SwitchConnectionProvider defaultSwitchConnProvider = switchConnectio
            .newInstance(defaultSwitchConnConfig);
```

7. Use Wiring in the Module

```
@Provides
@Singleton SwitchConnectionProviderList getOpenFlowJavaWiring(OpenFlowJavaWi
    return openFlowJavaWiring.getSwitchConnectionProviderList();
}
```

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

```
@Singleton
public class HwvtepSouthboundProvider implements Clus
    @Inject
    public HwvtepSouthboundProvider(@Reference final DataBroker
```

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

```
public class OvsdbModule extends AutoWiringModule {

    public OvsdbModule(GuiceClassPathBinder classPathBinder) {
        super(classPathBinder, "org.opendaylight.ovsdb");
    }

}
```

Thanks