Alt-datastores Proposal

Name

Alternative Data Stores.

Repo Name

alt-datastores

Description

Architectures

Single Adapter Architecture(Alignment with CDS)

Alt-datastores intends to centralize adapters of third-party databases like MongoDB, Etcd, PostgreSQL, CouchDB, etc.
An adapter would take either Binding Aware or Binding Independent data as input, and then do CRUD operations on the external database by calling the corresponding Java driver and leverage features of a third-party database such as clustering, sharding, transaction and so on, it implements DOMDataBroker and DataBroker of MD-SAL and provides these two services with a distinct type.
Applications could use these two services as an alternative choice instead of sal-distributed-datastore (CDS).
As for restconf, it should also reference DOMDataBroker and DOMDataTreeChangeService with an explict type which are provided by an adapter, then we can call rest api to access data in the third-party datastore.
In this way, one can run a lightweight standalone "ODL" process having a single global datastore for everything instead of CDS.

Here's an example framework of the datastore adapter, Alt-datastores provides default DOMDataBroker service instead of sal-distributed-datastore:

blocked URL

Multi-Adapter Architecture

Further, one application could use all of these third-party datastores at the same time if it wishes, then all supported database adapters are deployed together with alt-datastores, and users will configure, during deployment or even in runtime, which datastore to use. It requires alt-datastores to adapt restconf and data brokers, as a result, in the future, besides these 3rd-party databases, alt-datastores could also feed data to Kafka for applications of other systems, like ONAP DataLake, to consume, and even also it could help to replace a database.

blocked URL

Packaging and Migration

The exact technical design of the Packaging and Migration will need more analysis in the future, post project creation.

Adapters

Alt-datastores is a bridge between opendaylight and databases of the outside world, which makes opendaylight more flexible in a microservice architecture. Initially, alt-datastores would contain at least two adapters:

1. yongo
2. etcd

Yongo

Yongo, the yang-based adapter of MongoDB. https://github.com/opendaylight/alt-datastores/tree/master/mongodb

Deployment Requirement

Yongo is expected to support version of MongoDB 4.0 and above.
Yongo leverages transactions of MongoDB, so it's available for MongoDB's replica sets and sharded clusters only.

Design

Yongo creates two databases in MongoDB which are 'configuration' and 'operational' just like IMDS does, and every yang module is designed as a collection of MongoDB in which module's data is saved as an alone document.
For 'operational' database, yongo would cleanup all data everytime it starts while 'configuration' database holds data.

Below is a demo for compitabilty of Yongo, it provides DOMDataBroker and DataBroker services through blueprint with odl-type 'yongo' to be referenced. (In future, Yongo or Etcd as well as other adapters which implements DOMDataBrover correctly could replace the legacy service as default).

In 'yongo' blueprint.xml:

```
<service ref="yongoDOMDataBroker" interface="org.opendaylight.mdsal.dom.api.DOMDataBroker" odl:type="yongo"/>
<service ref="yongoBindingDataBroker" interface="org.opendaylight.mdsal.binding.api.DataBroker" odl:type="yongo"/>
```

BA application constructor injects DataBroker service of yongo:

```
   @Inject
   public ExampleImpl(@Reference(filter = "type=yongo") DataBroker yongoDataBroker) {
       this.dataBroker = yongoDataBroker;
   }
```

In 'restconf-nb-bierman02' blueprint.xml, set property value with the type 'yongo':

```
 <cm:property-placeholder persistent-id="org.opendaylight.restconf.databroker"
       placeholder-prefix = "@{" placeholder-suffix = "}" update-strategy="reload">
   <cm:default-properties>
     <cm:property name="databroker-service-type" value="yongo"/>
   </cm:default-properties>
 </cm:property-placeholder>
```

DataTreeChangeService

Yongo is designed to use change streams of MongoDB to implement it.
**Since MongoDB limits to notify the event of array elements deletion as an "update" type just with the array after deleting in 'updateDescription', yongo can't provide data changes notificatons with deleted data while the IMDS can. On the other hand, for new applications, it would be a choice that yongo provides a new publisher service just adapting to MongoDB's change streams.*
https://jira.mongodb.org/browse/SERVER-41559
Need more work...

Interactions with MD-SAL APIs

References

Yongo references DOMSchemaService to get runtime schema context for data converting.

Services

Yongo provides DOMDataBroker with odl-type 'yongo'.

Performance

Yongo is designed without using IMDS while Etcd does, as a result, we must have to read data from MongoDB every time which would be a bottleneck, on the other hand, for large data stores, it is also not appropriate to construct the whole data tree in memory through change events of a database, Yongo may be preferred to be usable in this case which is not caring the performance too much or one wants to use MongoDB in one way that already exists in a production environment.

- Testing Yang Model:

```
...
container test {
      presence "presence container";
      list outer-list {
          key id;
          leaf id {
              type uint16;
          }
         list inner-list {
              key name;
              leaf name {
                  type string;
              }
          }
      }
  ...
```

Synchronous

1.Benchmark of write transaction (using mongodb-driver-sync):

```
Benchmark                                                                        Mode   Cnt
Score      Error    Units
YongoBrokerWriteTxBenchmark.write2000SingleNodeWithOneInnerItemInCommitPerWriteBenchmark    avgt   10    4503.769
± 115.666  ms/op
YongoBrokerWriteTxBenchmark.write2000SingleNodeWithOneInnerItemInOneCommitBenchmark         avgt   10    9499.122
± 616.009  ms/op
YongoBrokerWriteTxBenchmark.write1000SingleNodeWithTenInnerItemsInCommitPerWriteBenchmark   avgt   10    1188.151
±  30.493  ms/op
YongoBrokerWriteTxBenchmark.write1000SingleNodeWithTenInnerItemsInOneCommitBenchmark        avgt   10    1180.524
±  23.827  ms/op
YongoBrokerWriteTxBenchmark.write500SingleNodeWithTwoInnerItemsInCommitPerWriteBenchmark    avgt   10    1905.055
±  68.027  ms/op
YongoBrokerWriteTxBenchmark.write500SingleNodeWithTwoInnerItemsInOneCommitBenchmark         avgt   10    2355.561
±  32.574  ms/op
```

2.Benchmark without actual write operation (performance consumed by the local computation):

```
Benchmark                                                                      Mode   Cnt
Score      Error   Units
YongoBrokerWriteTxBenchmark.write2000SingleNodeWithOneInnerItemInCommitPerWriteBenchmark   avgt   10    1320.513
± 27.634  ms/op
YongoBrokerWriteTxBenchmark.write2000SingleNodeWithOneInnerItemInOneCommitBenchmark        avgt   10     839.032
± 25.253  ms/op
YongoBrokerWriteTxBenchmark.write1000SingleNodeWithTwoInnerItemsInCommitPerWriteBenchmark  avgt   10     661.538
± 14.674  ms/op
YongoBrokerWriteTxBenchmark.write1000SingleNodeWithTwoInnerItemsInOneCommitBenchmark       avgt   10     433.262
± 19.941  ms/op
YongoBrokerWriteTxBenchmark.write500SingleNodeWithTenInnerItemsInCommitPerWriteBenchmark   avgt   10     360.239
± 13.131  ms/op
YongoBrokerWriteTxBenchmark.write500SingleNodeWithTenInnerItemsInOneCommitBenchmark        avgt   10     254.487
±  8.397  ms/op
```

Note:The oplog entry for the transaction must be within the BSON document size limit of 16MB.

3.Call tree from JProfiler:

- write2000SingleNodeWithOneInnerItemInCommitPerWriteBenchmark

blocked URL

- write2000SingleNodeWithOneInnerItemInOneCommitBenchmark

blocked URL

4.Conclusions:
Performence of commit per write is better than in one commit, MongoDB consumes most of the performance.

Asynchronous

Benchmark of using mongodb-driver-reactivestreams:
Need more work...

Data Access

Yongo saves yang modeled data to the MongoDB, so not only odl but also other systems could access directly through yang schema path by adapting corresponding driver APIs of MongoDB a bit.

blocked URL

A general overview of netconf project:
1.Sequence Diagram
2.Component Diagram

Etcd

This adapter was previously created by Michael Vorburger, and is currently available GitHub at https://github.com/vorburger/opendaylight-etcd. We propose to "seed" (initial import, preserve git log) the Git repo of this new project with the code from that https://github.com/vorburger/opendaylight-etcd.

Scope

1. Alt-datastores would include adapters of 3rd-party datastores, one adapter like Yongo should at least have ability of saving binding/normorlized data to a 3rd-party datastore;
2. Alt-datastores would be an alternative to CDS to provide other DOMDataBroker services that is transparent to other components.
3. Multiple adapters could be applied for replica data purpose.

Resources Committed (developers committed to working)

See below for who is.

Initial Committers

- han.jie@zte.com.cn (Jie Han)
- vorburger@redhat.com (Michael Vorburger)
- xiong.quan@zte.com.cn (Xiong Quan)
- lijiansong@chinatelecom.cn (Li Jiansong)
- guobiaomo@chinamobile.com (Guobiao Mo)
- zhangminyj@chinamobile.com (Zhang Min)
- jinkaiwen@chinamobile.com (Jin Kaiwen)

Vendor Neutral

The project is made from seed Etcd which is based on initial public code from a long time contributor.

No other vendor code, logos nor is anything included.

Meets Board Policy (including IPR)

New Project. Inbound Code Review of both the initial opendaylight-etcd seed and the subsequent yongo (MongoDB) implementation will scanned for IPR (email to Steven Winslow).

opendaylight-etcd depends mostly on org.opendaylight code (mdsal, controller, infrautils), plus the usual suspects of 3rd-party libraries which are already widely used in ODL (e.g. Guava, Netty, Maven tooling), and the etcd Java client jetcd from https://github.com/etcd-io/jetcd, which is Apache licensed (and on which Michael Vorburger is also a committer).