Data Validation using Commit Cohort

ODL Magnesium DDF, Antwerp - September 2019

Deepthi V V, Lumina Networks

Agenda

- Why is it required
- How does it work
- Generic framework for Apps
- Usage Considerations

Why is it required

- In OpenDaylight, structure of data is defined using YANG models
- Controller performs syntactic validation on data written to data-tree

```
<error-message>Error parsing input: Schema for node with name udp-only and namespace
urn:opendaylight:netconf-node-topology does not exist at
AbsoluteSchemaPath{path=[(urn:TBD:params:xml:ns:yang:network-topology?revision=2013-10-21)network-topology,
(urn:TBD:params:xml:ns:yang:network-topology?revision=2013-10-21)topology,
(urn:TBD:params:xml:ns:yang:network-topology?revision=2013-10-21)node]}</error-message>

<error-message>Error parsing input: Schema node with name fb-port-id was not found under
(urn:TBD:params:xml:ns:yang:network-topology?revision=2013-10-21)termination-point.</error-message>
```

Why is it required

- Application business logic may require additional semantic validations
- Semantically invalid data may get written into data-tree
- Adds complexity to App for keeping DS contents and App state in-sync
- Semantic validation prior to data getting updated in data-tree can avoid this

Why is it required - Examples

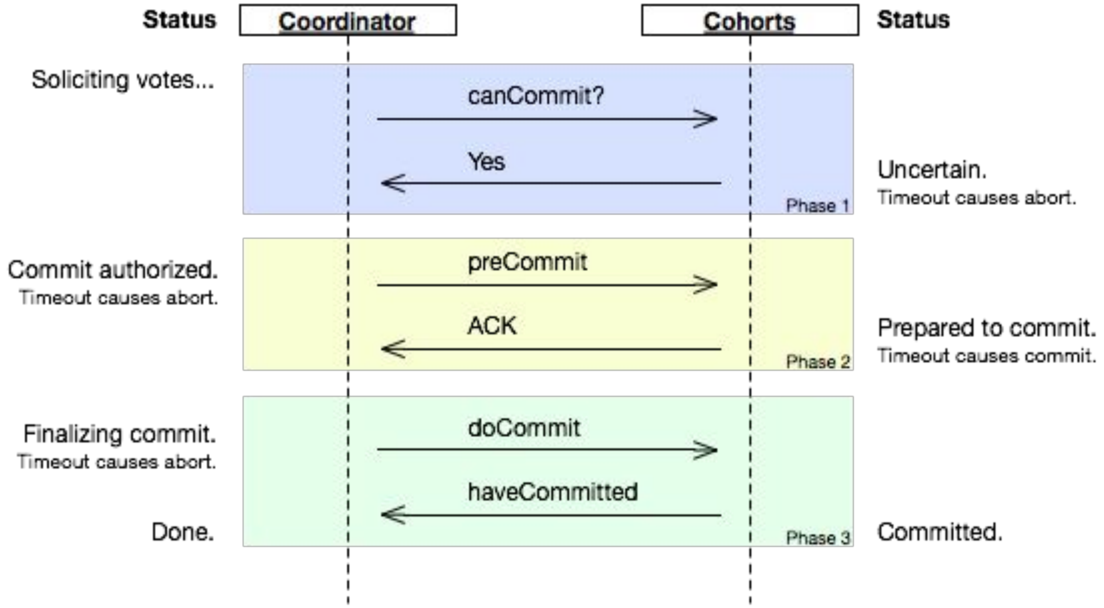
- Data dependency validation
  - Inventory model defines details of network elements (NEs), service model defines network services that can be created in network
  - Service instance refers to elements from inventory data
  - Inventory data used by service instance MUST be present
  - Ten NEs are present in inventory (n1..n10)
  - Service instance (s1) created which uses n1, n5
  - Request received to delete n5
  - If n5 is deleted, s1 should be unprovisioned and marked as in-complete
  - Do not allow deletion of n5 as it is used by s1

- Validate correctness or completeness of data
  - Certain fields cannot be modified

- Deny changes to config till App is ready

How does it work

- OpenDaylight data-tree write transaction follows three-phase commit protocol (3PC)



How does it work

- Using `DOMDataTreeCommitCohortRegistry`, a commit cohort (`DOMDataTreeCommitCohort`) can be registered for a data-tree path (`DOMDataTreeIdentifier)`

- The registered commit cohort will participate in 3PC involving data-tree modifications at given path

- Commit cohort can validate data-tree modifications, with option of rejecting the supplied modification

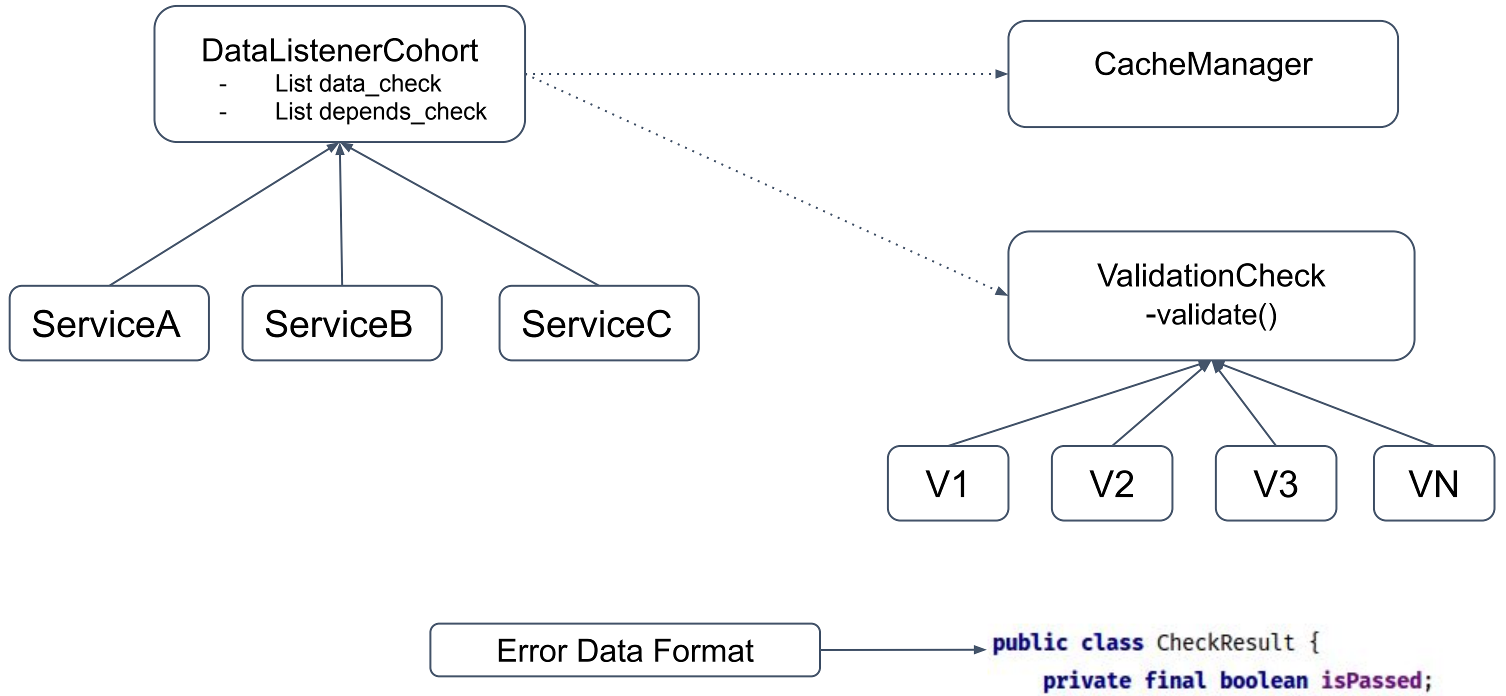
- Rejection is signalled by throwing `DataValidationFailedException` from implemented `canCommit()` method

- Since this works at Tx level, it is agnostic to how the modified was triggered (RESTCONF or Java API)

Generic framework for Apps

- Makes it easy for Apps to perform semantic validations
- Built on top of MDSAL commit cohort API
- Data caching is used to enable data dependency checks
- Enables proper error message to be reported for failed check

Generic framework for Apps



```
public class CheckResult {
    private final boolean isPassed;
    private final String errorCode;
    private final String errorMessage;
```

Usage considerations

- Validation must be done FAST as it holds-up the progress of Tx
  - Advisable not depend on any external resources

- Callback MUST NOT use any data Tx APIs
  - Any other data needed for validation (e.g. data dependency check) should be cached

Thanks