ONOS ISSU Protocol

September, 2017

What we’re talking about today

- Objectives
- Multi-version clusters in “Loon”
- Limitations of multi-version clusters
- A new upgrade protocol
- Overcoming current limitations
- Timeline for release

The Loon Release

Objectives

- Support In-Service Software Upgrade for the ONOS core and applications *without interrupting service*
- Focus of the Loon release:
  - Foundational work to support clusters running multiple versions of the software during upgrades
  - Stores communicate with and persist Java objects, these can evolve across versions
  - Backward/forward compatibility required for upgraded nodes to participate in consensus during upgrades

Loon deliverables

- Extended Kryo to support compatible serialization
  - Defaults to using CompatibleFieldSerializer if onos.cluster.issu.enabled property is true (false as default in Loon release)
- Refactored all primitive storage (Raft protocol, logs, etc) to support Kryo serialization
- Added versioning to Raft logs and other config files to support backwards compatibility

Limitations

- Multi-version clusters don't account for changes in storage patterns, data structures, protocols, and other logic
  - Different primitives across versions of a store
  - Changes to partitioning schemes within primitives
  - Conflicting flows across application versions
- No mechanism to prevent incompatible upgrades
- No mechanism for easily rolling back failed upgrades

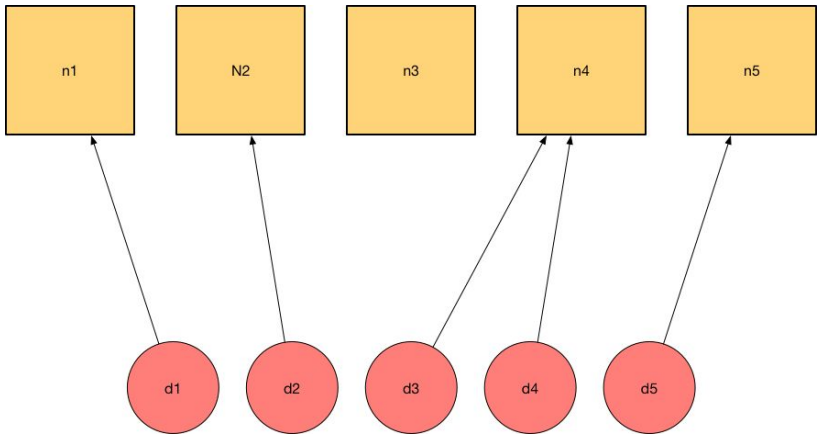
Protocol requirements

- Isolate store/application state/communication across versions
- Support transforming/migrating existing state during upgrades
- Provide mechanism for rolling back failed upgrades
- Preserve fault tolerance guarantees during upgrades

ISSU Protocol

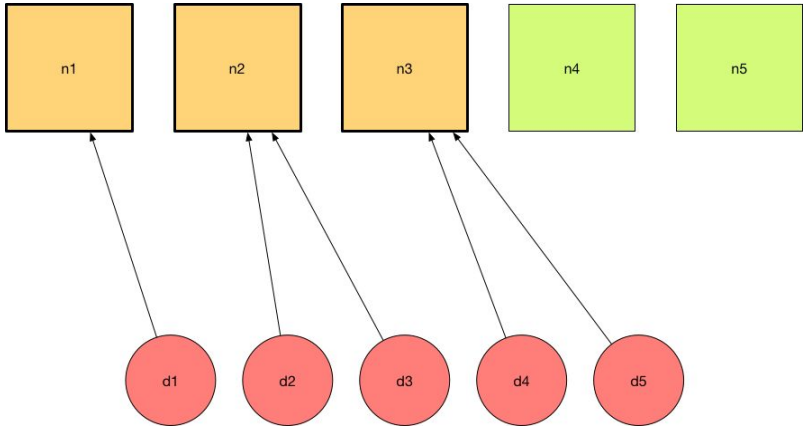
Initialize the ISSU protocol

- Run “issu init”
- Sets most stores to read only mode



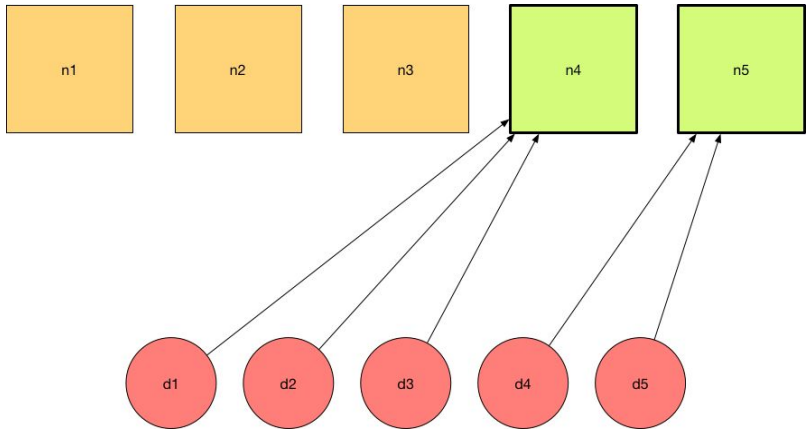
Upgrade a subset of the cluster

- A minority of nodes are upgraded to preserve fault tolerance
- Mastership is reassigned to *old* nodes
- Upgraded nodes are initialized with snapshots of the cluster's state when the upgrade was initialized
- State is isolated within each version



Run the upgrade

- Run “issu upgrade” command
- Switches mastership from the old subset of the cluster to the new subset

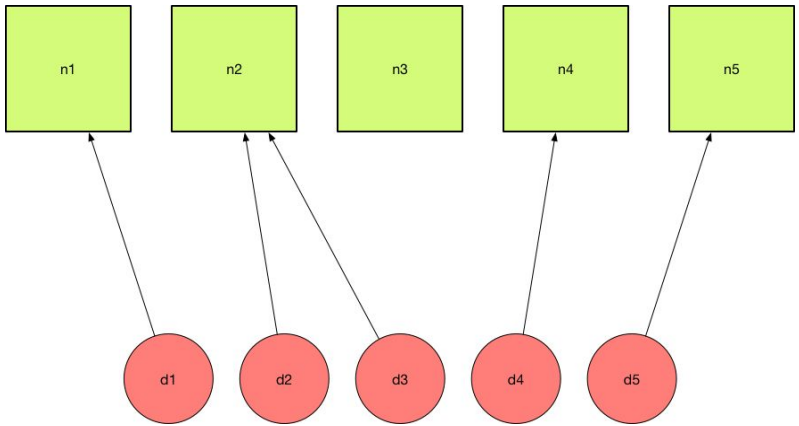


Verify the upgrade

- Log in to the upgraded subset of the cluster and verify the state and operation of the cluster
- May be partially automated in the future

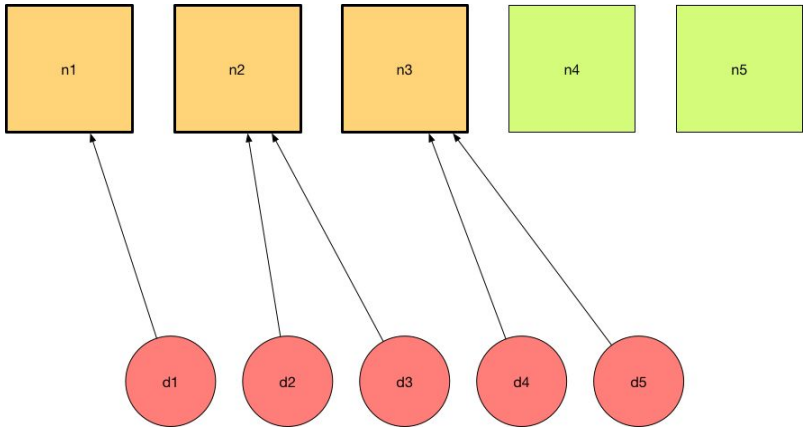
Commit the upgrade

- Once the upgrade is verified, upgrade remaining nodes
- Commit the upgrade with “issu commit”
- Mastership is rebalanced among upgraded nodes
- *v-old* state is purged from the system



Rollback a failed upgrade

- If upgrade verification fails, run “issu rollback” to roll back the upgrade
- Mastership is reassigned to old nodes
- Write locks on all stores are released



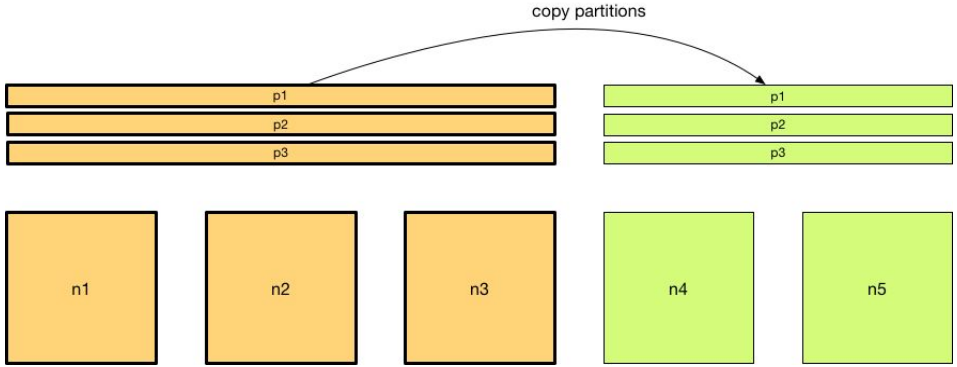
Upgrade Manager

- Coordinates the upgrade process (supports the upgrade commands)
- In-process service running locally on each controller node
- Uses consistent primitives backed by a Raft partition that spans the entire cluster

Overcoming Limitations

Isolation

- *v-new* nodes copy and fork partitions from *v-old* nodes by joining the existing Raft partitions as non-voting members
- *v-new* nodes create versioned Raft partitions from snapshots of old partitions and form a new cluster
- Communication is isolated for each version of the cluster
- Read-only mode is used to prevent divergence of state between old and new subsets



State migration

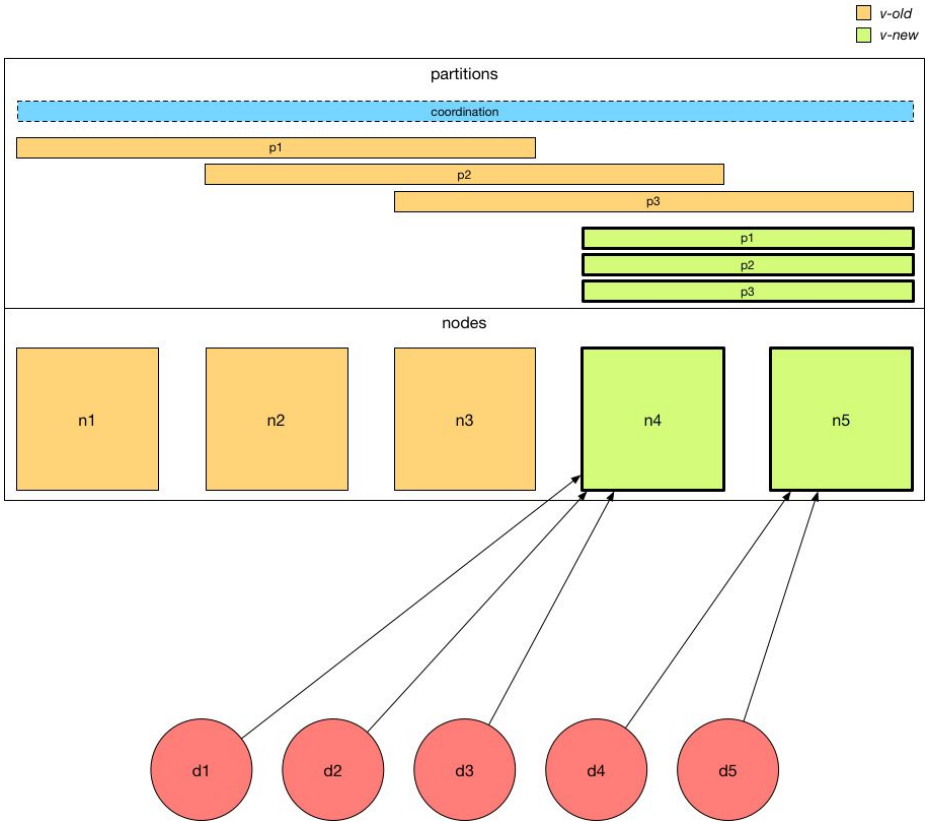
- State can be modified by upgraded nodes without affecting the ability to roll back upgrades
- Store/application initializers read from the snapshot of old state and modify it at startup
- Reduces the complexity of managing offline upgrade paths by using built-in mechanisms to upgrade state in a deterministic manner

Upgrade rollbacks

- Preserving old state in snapshots ensures state can be easily restored on rollbacks
- Using mastership changes provides a mechanism to instantaneously activate and roll back upgrades
- Upgrading controller nodes in place ensures upgrades and rollbacks can be done without changes to device configurations

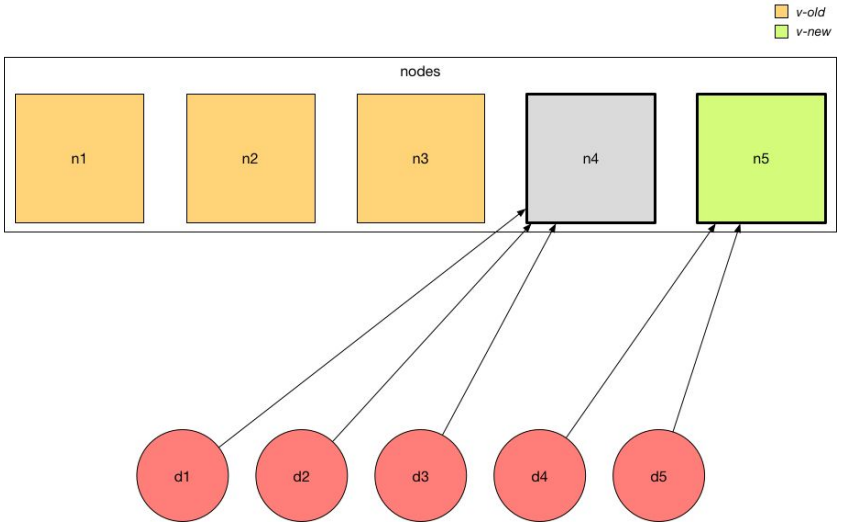
Fault tolerance

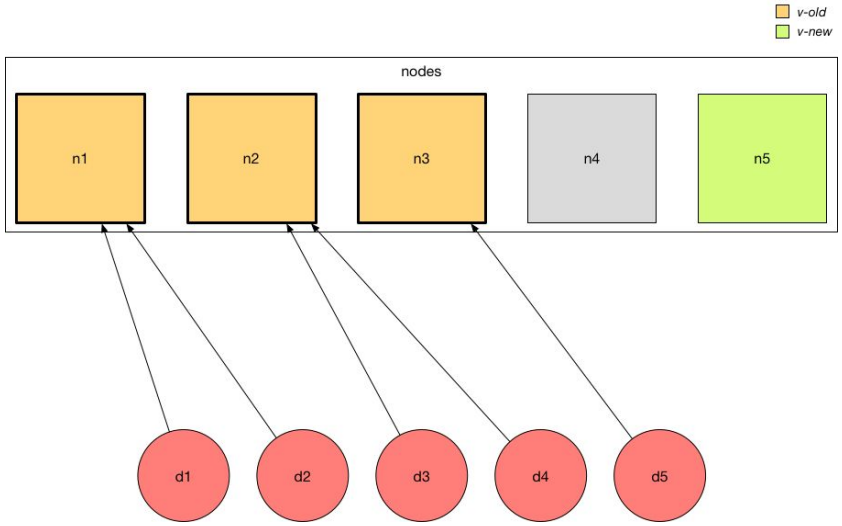
Backwards compatibility for consensus is ensured by allowing the consensus protocol to continue to run on all nodes of both versions on behalf of the `v-old` subset of the cluster during the upgrade to maintain fault tolerance for the controller overall



Fault tolerance

Upgraded nodes are made fault tolerant during upgrades through failure detection, mastership reassignment to the *v-old* nodes, and automatic rollbacks





Timeline

Timeline

- “Loon” release
  - Revision upgrades via multi-version clusters
- “Magpie” release
  - ISSU commands
  - Mastership based upgrades
  - State isolation
- “N” release
  - Fault tolerance
  - Automated rollbacks
  - Rollback timers

ISSU Brigade

- First meeting will be held immediately after ONOS Build
- For more information:
  - Wiki: https://wiki.onosproject.org/display/ONOS/ISSU
  - Google Group: https://groups.google.com/a/onosproject.org/forum/#!forum/brigade-issu
  - Slack Channel: #brigade-issu

Software Defined Transformation of Service Provider Networks

Join the journey @ onosproject.org