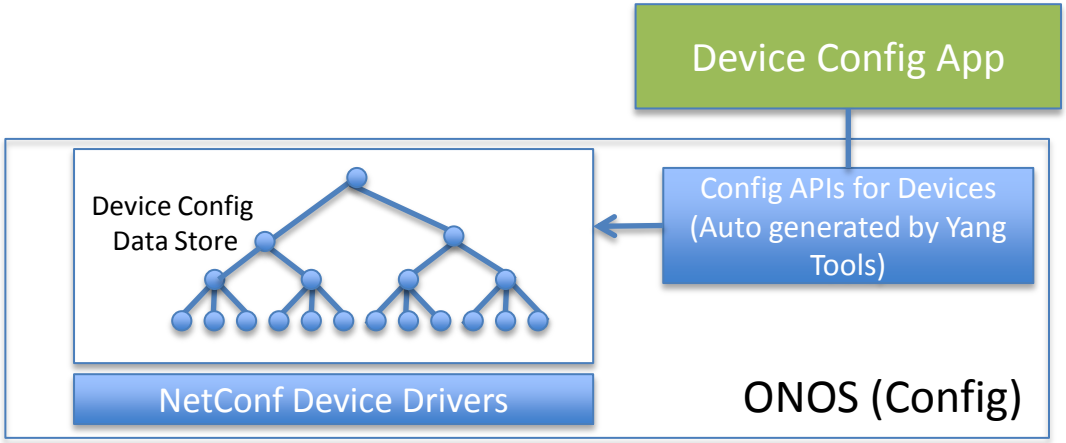
Dynamic Configuration and Provisioning of Devices and Services with ONOS

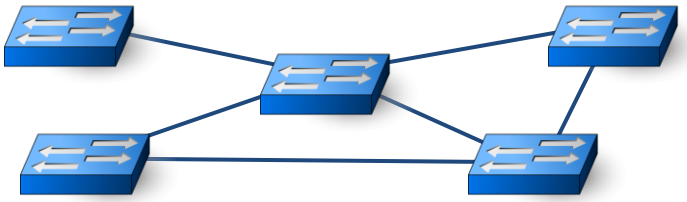
Brigade Dynamic Configuration
Nov, 2016

Dynamic Configuration of Devices

- Goal: Enable a network operator to seamlessly bring up/down and configure devices from different vendors and to verify the config
  - With minimal or no human intervention
- Benefits
  - Network operators: Significant opex savings and vendor independence
  - Vendors: Faster integration of its products into operators' networks and better value prop to its customers (e.g., reduced opex)

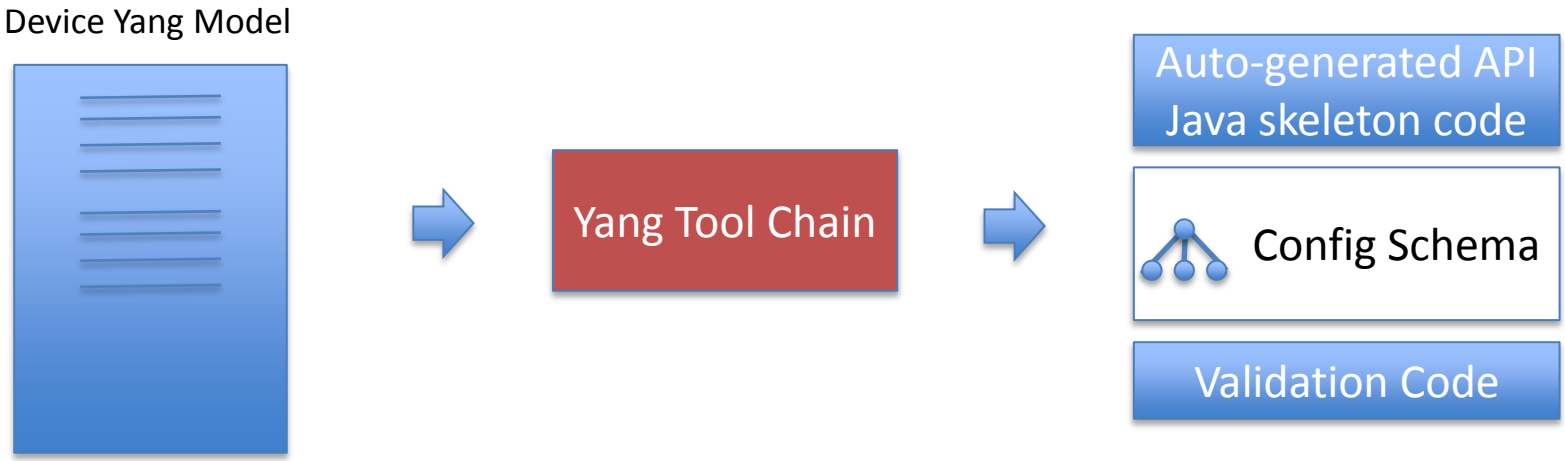
Dynamic Configuration of Devices





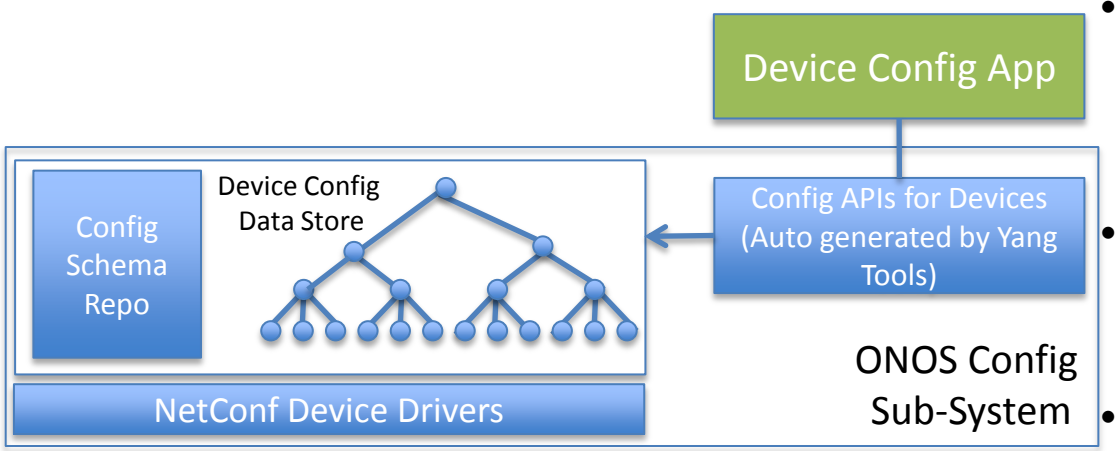
- ONOS solution based on NetConf and Yang –industry standards
- The Device Config Store contains all device specific config info
  - Schema and actual config parameters
- The Device Config APIs are auto generated by Yang tool chain and device Yang model
- A device config app uses auto generated API to configure a given device
  - The API writes the config parameters into the store
  - The store generates a notification to the driver software to actually program the device via NetConf

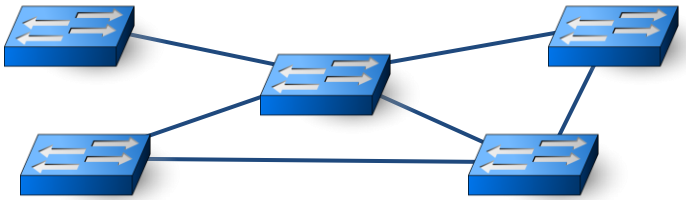
Yang Tool Chain (for ONOS)



- Creation of Yang Tool Chain a significant milestone
  - Could not use ODL Yang Tool Chain as it is too tied to MD-SAL
  - It is difficult to make it completely platform independent
    - Ours is tied to ONOS APIs
- For every device type, the vendor provides the Yang model which is used to auto create a schema, skeleton code and validation code: the schema is stored in a repository and the code with the API is added to the code base
- A developer fills in the details in the skeleton code (the device specific logic) to make Java code complete

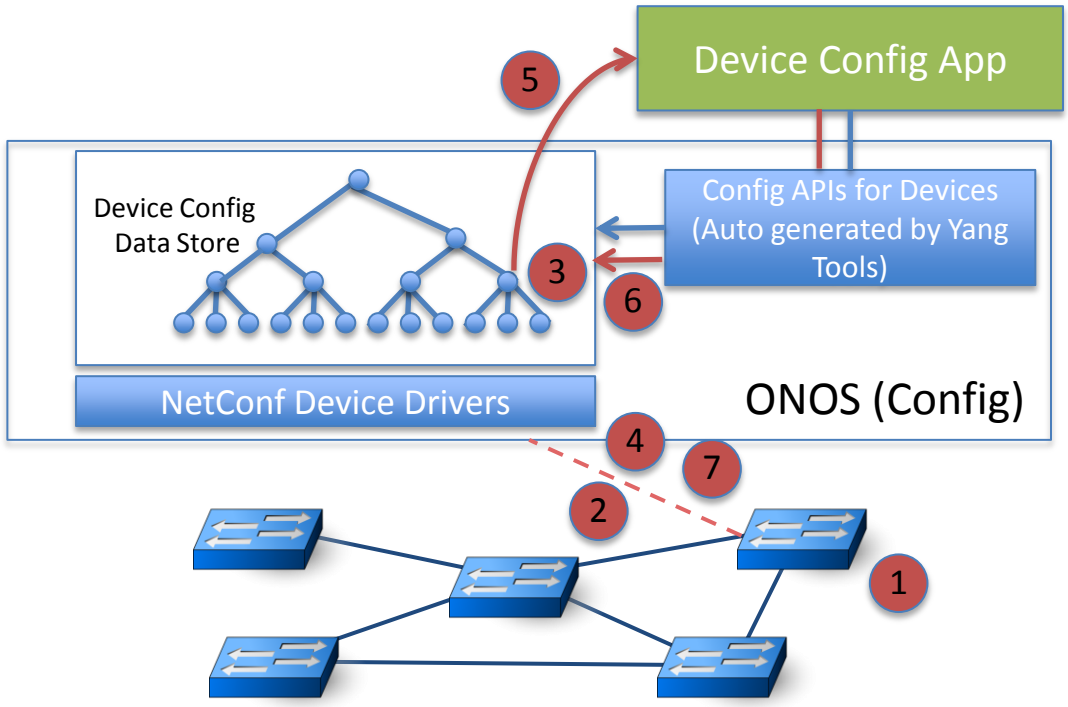
Dynamic Configuration of Devices





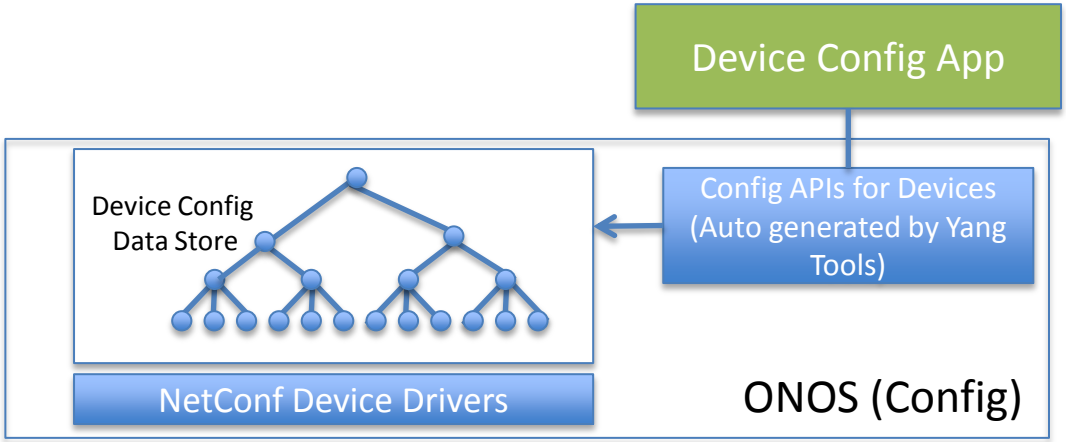
- When a device is connected to the net, ONOS auto discovers it
  - Devices boot off of the network (similar to servers & switches in a datacenter)
- Using the device config schema, an instance of the device config is created in the config tree (store)
- Using NetConf/Yang the device initializes its config instance in the config tree
- Config tree (store) can notify the app listening for an update to further config the device using the Config API

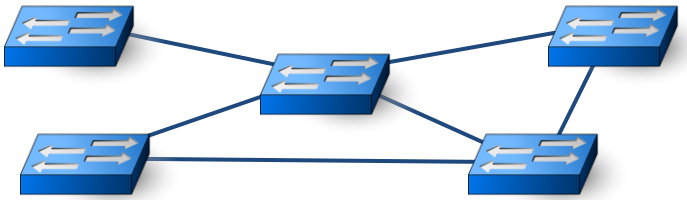
Dynamic Configuration of Devices: Example



1. Device is connected to the net
2. ONOS auto discovers it
3. Device Config instance is created in Config Store
4. Using NetConf/Yang the device initializes its configuration in the config store
5. Config store notifies an app
6. The app further configures the device
7. Device drivers actually write the config into the device

Scalability and Performance Challenges

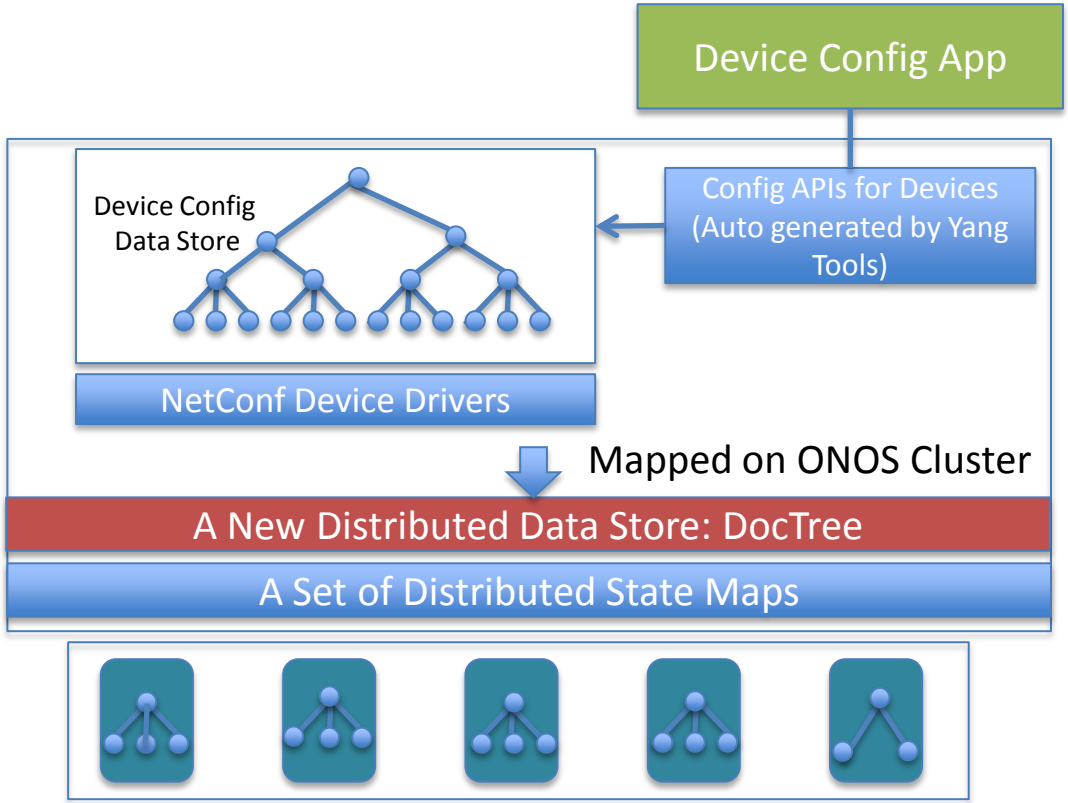




- A network of a large service provider may include 100K+ devices and 100M+ ports
  - Lot of dynamism: devices go up/down for service upgrades as well as due to failures; new services being provisioned all the time
- The Config Data Store implementation is a challenge
  - 100M+ nodes in the tree
  - Need prefix based search operations
  - Want transaction semantics
  - O(1K) device config operations per second
  - O(1K) service config operations per second
  - Peaks maybe much larger

  With HA and durability

Distributed Implementation of Config Data Store with ONOS Clustering



- ONOS is horizontally scalable and uses a number of servers as a cluster
- ONOS supports a variety of distributed data stores with different consistency, availability and durability attributes
- A new distributed data store called DocTree has be created to implement Device Config Data Store on the ONOS cluster
- The DocTree data store
  - Takes advantage of ONOS cluster
  - Efficiently implements Config Tree
  - Allows efficient prefix based searches
  - Allows transaction semantics
  - Allows HA and durability as appropriate

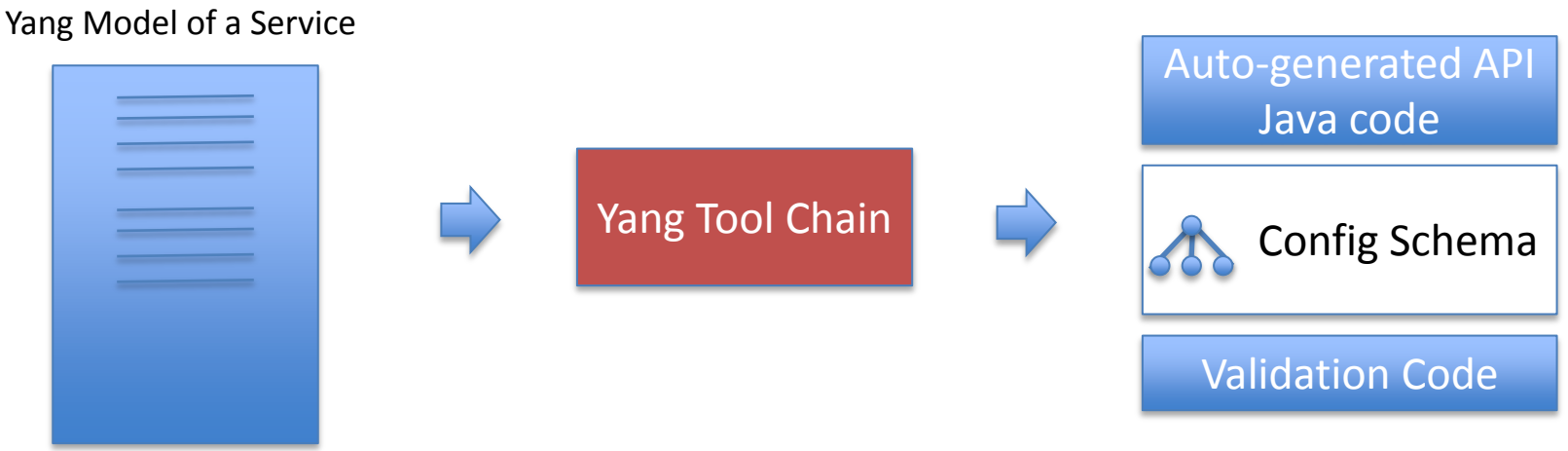
Don't know if any other solution can do this

How about dynamic Configuration and Provisioning of Services?

Dynamic Configuration/Provisioning of Services

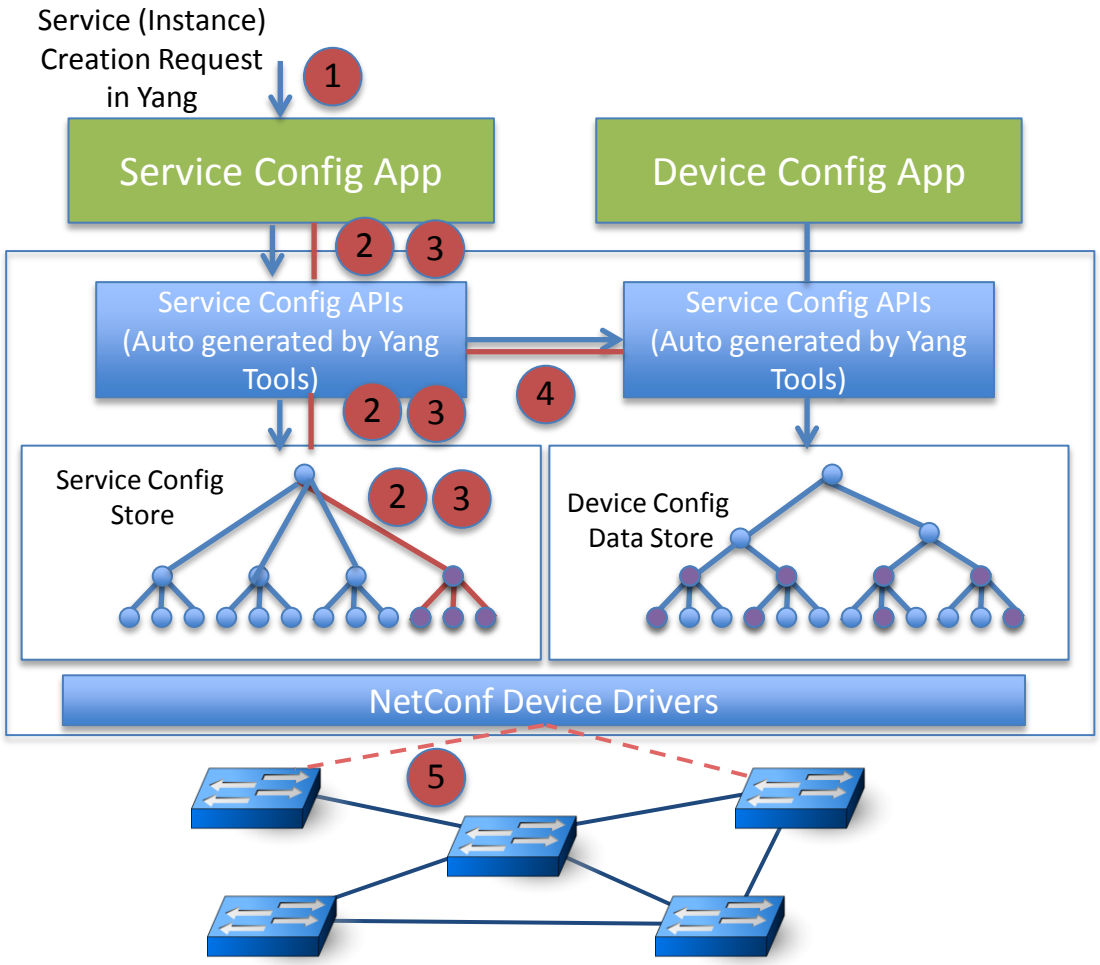
- Goal: Enable a network operator to seamlessly configure and provision a service on the network comprising many devices from many vendors
  - With minimal or no human intervention
- Benefits
  - Network operators: Agility to deploy new services with reduced opex
  - Vendors: Opportunity to support many services on the devices

Service Models in Yang



- A new service creator writes a Yang Model
- Yang Tool Chain translates the Yang Model of the Service into (1) API calls, skeleton Java code, a config schema, and some validation code
- The service creator also write business logic into the skeleton code to “implement” service provisioning based on the underlying platform such as ONOS
- Service schema is added to a schema repository; code is added to the code base

Provisioning of a Service: Create an Instance



1. A request is received to instantiate a service (already specified before)
   - Its schema is in Schema Repo and code in code base with APIs auto generated
2. The Service Config App adds an instance of service config to the service config tree (store)
3. The Config App programs the instance specific config parameters into config tree
4. The Config App (or something) will initiate writing of new config parameters in Device Config Tree
5. Device drivers actually write the new config parameters into the device