OpenDaylight Scaling Architecture

Presenters:
Konstantinos Kanonakis
Manoj Chokka

01/12/2022

Agenda

- **Goal of the presentation**
- **Verizon's ODL scaling requirements**
- **Existing ODL scaling options and issues**
- **Verizon's proposed architecture**
- **Q&A**

Goal of the Presentation

- Introduce an alternative microservices based ODL scaling architecture
- Present a high-level overview of the architecture
- Get feedback from the community
- Improve architecture further along with the community

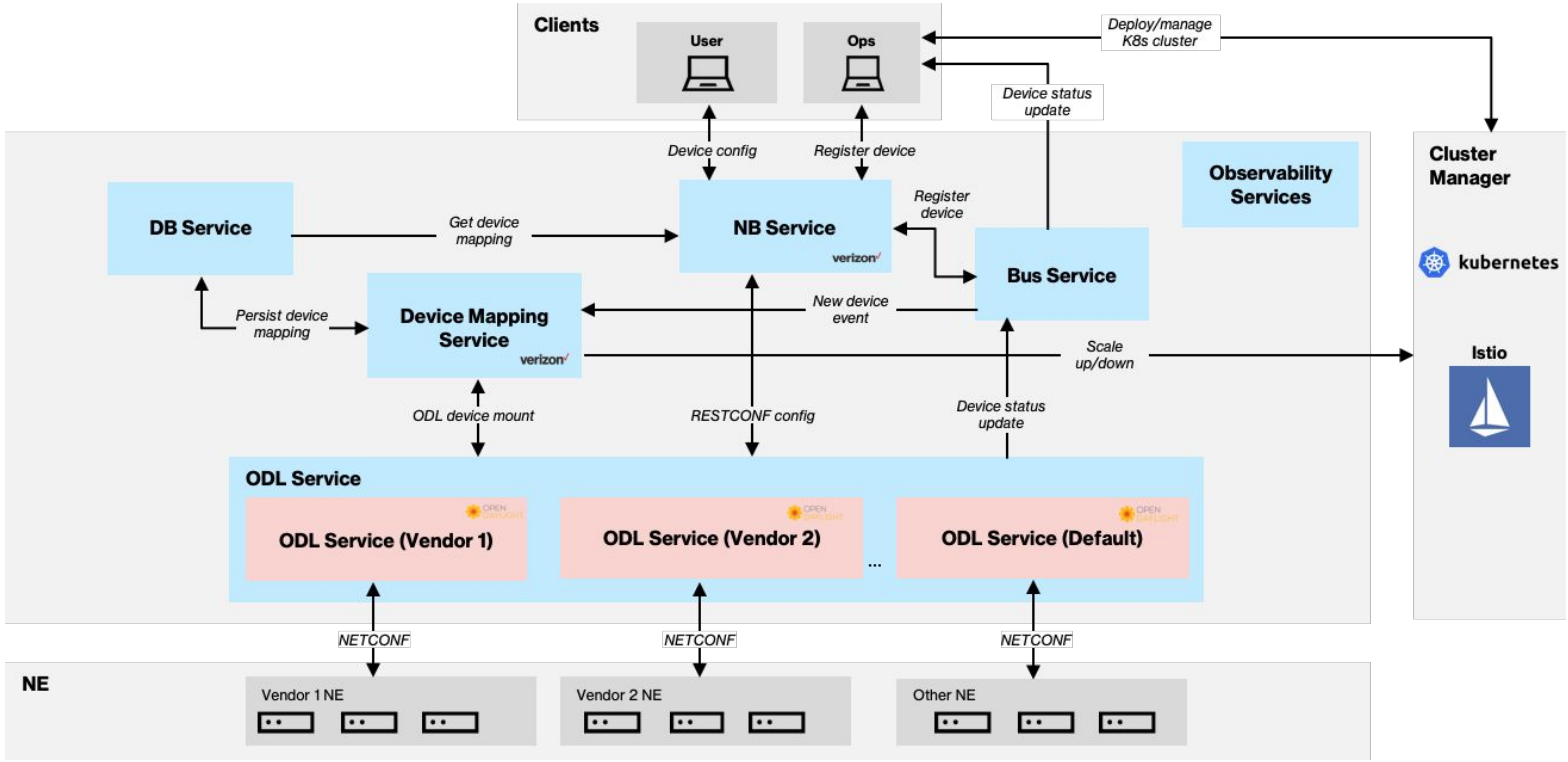
Verizon's ODL Scaling Requirements

- Manage connections to 500K+ devices in production
- Different memory and CPU requirements of devices
- Make ODL Highly Available (HA)
- Keep ODL stateless
- Microservice/container-based deployment

Existing ODL Scaling Options and Issues

- Current ODL uses AKKA cluster
  - Data sharding to exchange the information between ODL instances
- ODL-Micro based NETCONF and OpenFlow distributions available for container-based deployments
  - But complete scaling framework not available.
- Other solutions are not microservices-compatible

Verizon's Proposed Architecture



Core Architecture Components

Northbound (NB) Service

- Handles external device registration and configuration requests
- Uses ODL RESTCONF APIs to apply device configuration

OpenDaylight (ODL) Service

- Includes containerized stateless ODL with NETCONF/RESTCONF functionality only
- Includes custom Sidecar container for monitoring device connectivity

Device Mapping Service

- Assigns each device to ODL Pods

Database (DB) Service

- External (non-ODL) DB storing device metadata and mapping of devices to ODL instances (Pods)

Bus Service

- Relays async events/notifications among services

Observability and Metrics

Tracing

- Istio provides the essentials to trace interactions between micro-services

Logging

- Logs from ODL Pods and other applications pushed to the Logging Service

Metrics

- Sidecar container in ODL Service collects statistics from ODL Pods and stores them in the Metrics Service

Device Registration

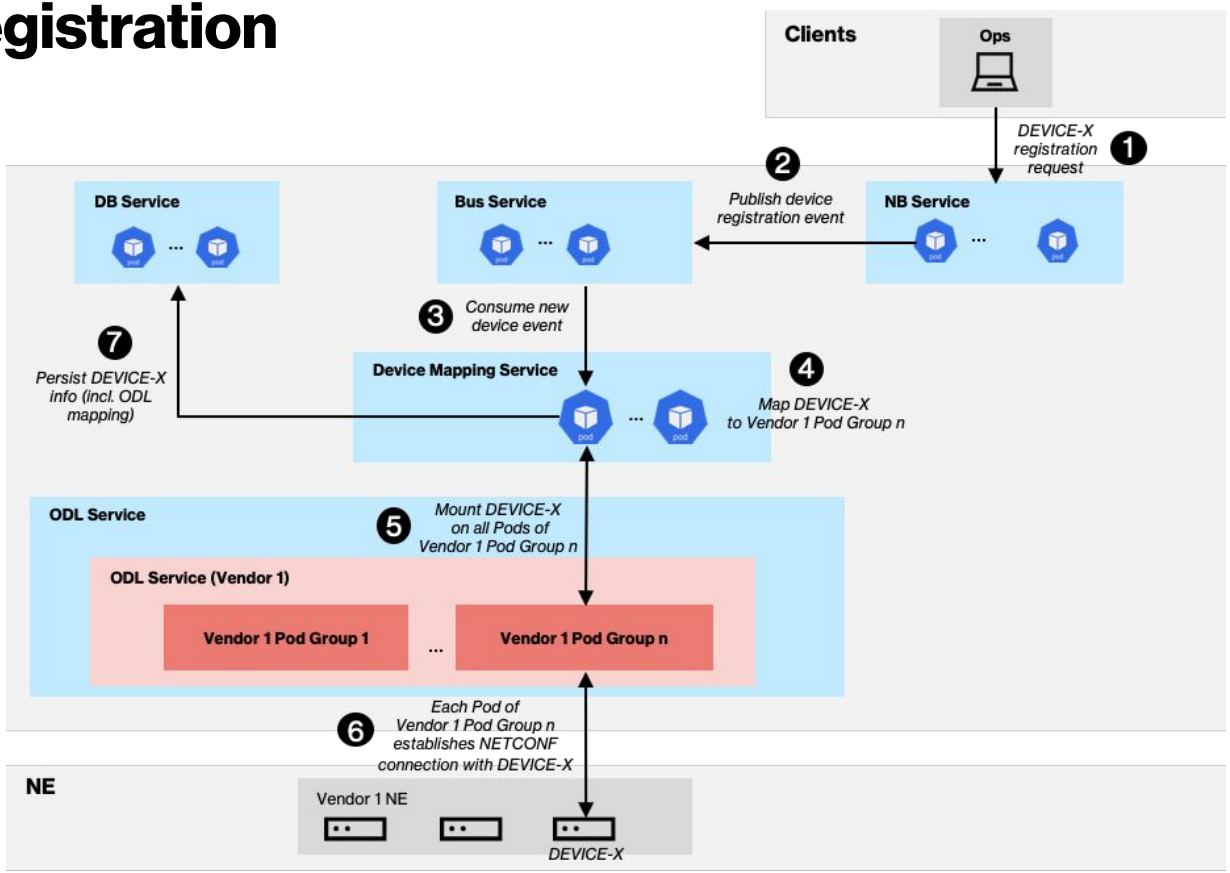
ODL Pod Discovery

- Device Mapping Service monitors K8s registry for new ODL Pods
- When new ODL Pod is discovered, it creates an entry in the DB including its IP address, zone, K8s namespace
- If HA is required then Pods should be in different K8s zones

Device Registration Process

- NB device registration request is made including device metadata and NETCONF connection parameters
- Device Mapper picks up event via the Bus and maps device to ODL Pods in the ODL Service of that vendor
    - Placement algorithm may depend on # of devices mounted on each instance, avg. response times etc.
    - Can be modified for different use cases
- Device Mapper mounts the device on each of the selected ODL Pods using their actual IPs
- Device Mapper persists the device metadata and ODL Pod mapping in the DB

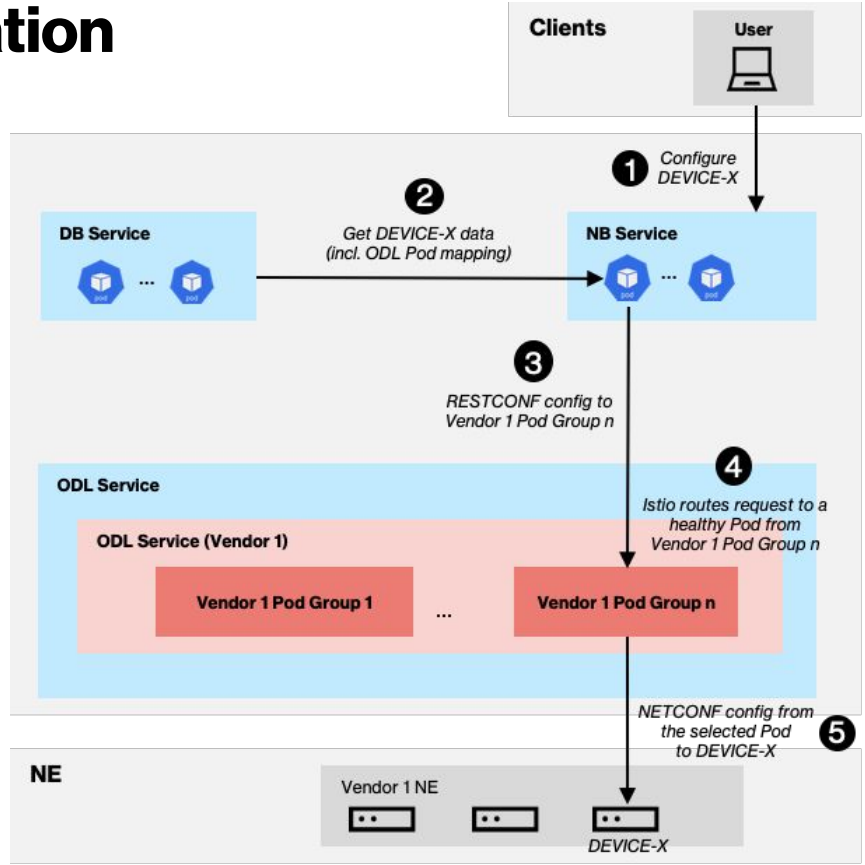
Device Registration



Device Configuration

- NB device configuration request made to the NB Service including device name along with config parameters
- NB Service retrieves from DB the DNS name of ODL Service managing the device and HTTP connection params
  - Note: HTTP params could be part of the K8s deployment if common across ODL instances
- If device not registered or not mounted on any ODL Pod, then a NB error is returned
- Otherwise, NB Service makes RESTCONF device config request using as server name the ODL Service DNS name
  - For example: *https://**odl-vendor1-2**.svc.cluster.local:18181/rests/data/...*
- Istio selects one healthy Pod of the particular ODL Service
- ODL instance receiving request applies corresponding configuration to device using NETCONF

Device Configuration



ODL Pod Failure

Pod Failure

- Istio detects Pod failure via health checks and marks Pod as unhealthy
- Istio ensures that NB Service can still communicate with the devices as long as there is still one healthy Pod per device
- The K8s infra ensures that Pods are eventually brought up again

ODL Pod Recovery

- Device Mapping Service detects Pod recovery by monitoring ODL Pod health
- Identifies devices managed by recovered Pod and mounts them on the corresponding ODL instance

Consistently Failing ODL Pods

- Infrastructure issues (e.g. corrupted VM) may prevent ODL Pods from recovering using scheduled K8s Nodes
- If Pod keeps failing, Device Mapping adds “replacement” Pod (same namespace and zone) to K8s configuration
- Device Mapping Service identifies all devices managed by failing Pod and reassigns them to Pods not including it

Device Connectivity Failure

Detecting Device Disconnection

- ODL Sidecar subscribes to NETCONF notifications from collocated ODL instance and detects device mounting status change
- Sidecar publishes event on Bus Service and Device Mapping Services receives it

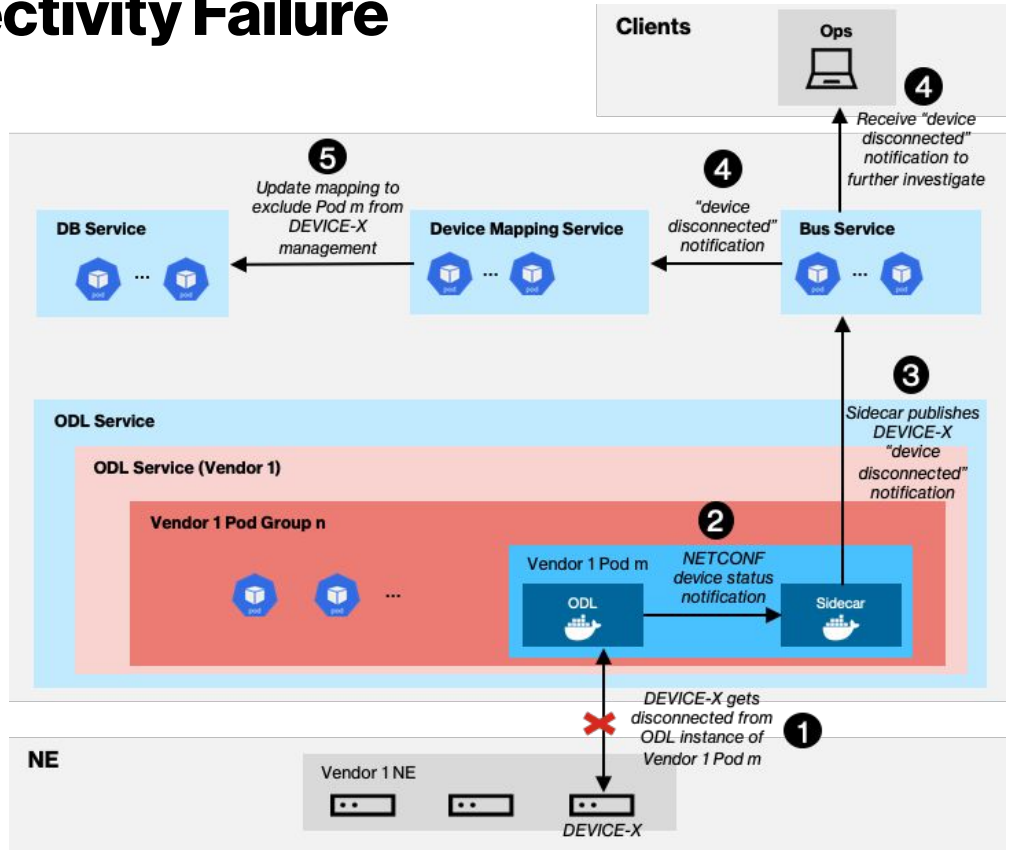
Device Disconnection Handling

- Device Mapper updates device mapping in DB to exclude Pod from which the device is still disconnected
- This ensures that Istio will never route device configuration API calls towards Pods to which device is not connected

Device Re-Connection Handling

- The reverse process is followed (Sidecar detects status change, Device Mapping Service updates device status/Pod assignment)

Device Connectivity Failure



Scaling the ODL Service

Device Mapper Configuration

- Desired metric values (e.g. Pod CPU/Memory usage, devices per Pod)
- Equations calculating desired # of ODL Pod replicas based on current replicas and current, desired metric values

Scaling Up

- When Device Mapper calculates desired # of replicas higher than current #, it updates K8s configuration to include additional ODL Pod pairs
- For new devices, Device Mapper selects ODL Pods from the pool (which now includes the newly deployed Pods)

Scaling Down

- When Device Mapper calculates desired # of replicas lower than current # for a certain period of time (e.g. five minutes):
- Identifies the ODL Pods that should be removed and which devices are currently mapped to them
- Re-assigns each of the affected devices to a Pod pair not including the removed Pods, performing mounting as needed
- Updates the K8s configuration to removing the unneeded amount ODL Pods

Using a K8s Horizontal Pod Autoscaler (HPA)

- Possible, but limited flexibility (e.g. need for deploying Pods in pairs) and increased downtime (Pod removals have to be detected)

Further Steps

- The presented architecture can benefit from:
  - ODL-Micro
    - Needs improvements to make it production ready
  - ODL Helm project
    - Karaf-based Docker image already available for NETCONF plugin
    - Helm chart work is in progress
    - Identify if/how the Helm charts can work with the architecture
- Community can contribute to either the architecture or the individual components and strengthen it wherever necessary
- We would be interested to discuss this further within the community

Q&A