Experience. Outcomes.

VOLTHA Architecture

V2.0

Sergio Slobodrian

Wednesday December 5, 2018

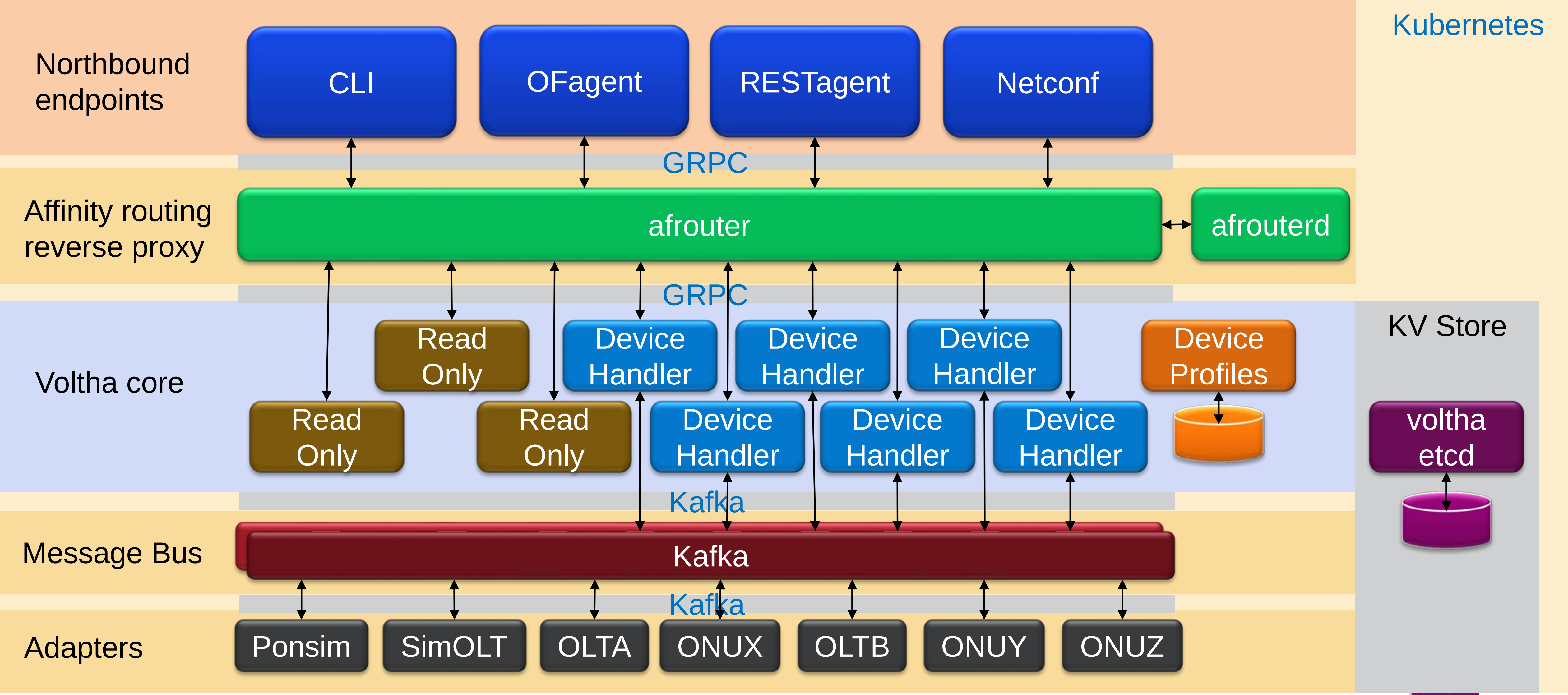
Contents

- High Level Architecture
- High Availability Model
- Kafka Adapter Messaging Model
- Per µService Architectures
- Call flows through the system
- Kubernetes Integration

Subhead Information

High Level Architecture

High Level Architecture

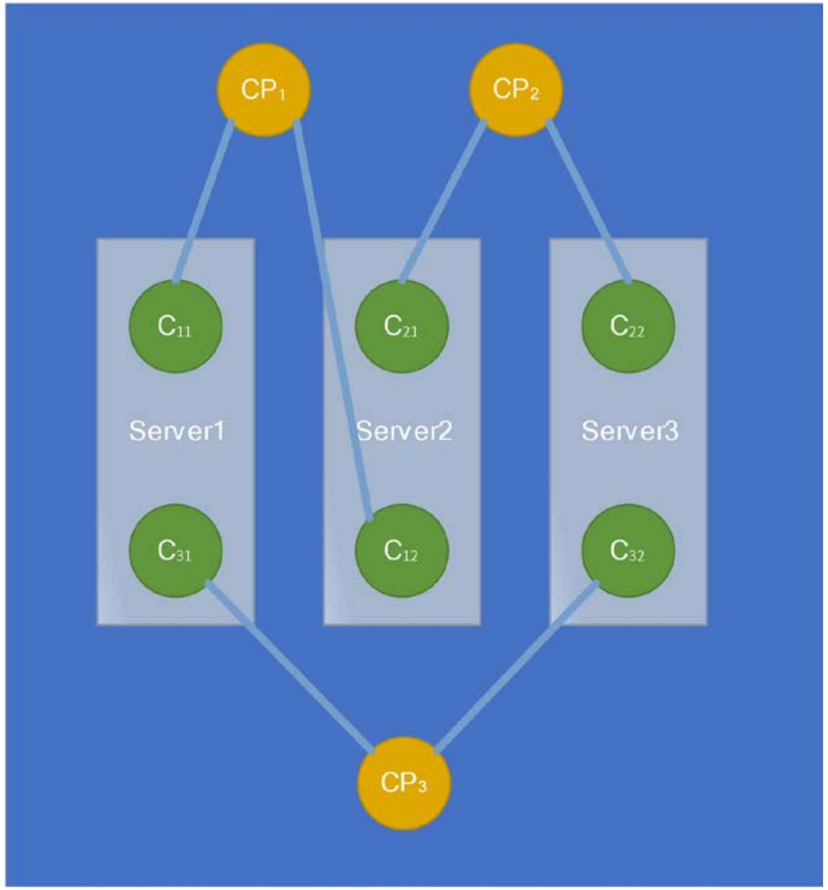


Subhead Information

High availability model

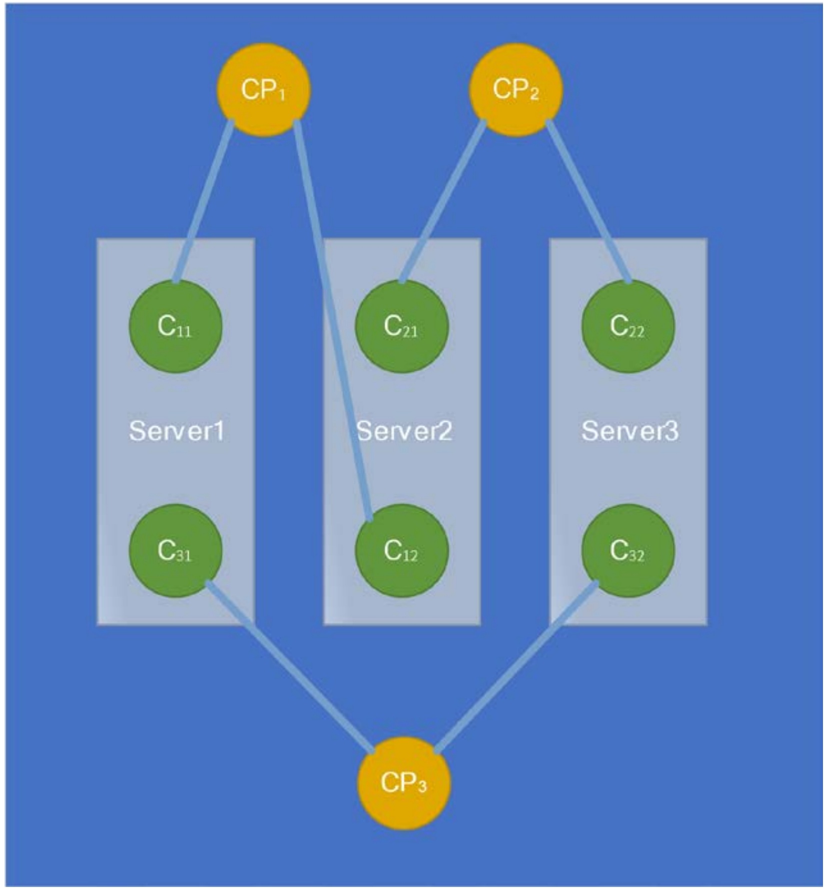
High Availability Model

- **Device handlers are arranged in active/active pairs.**



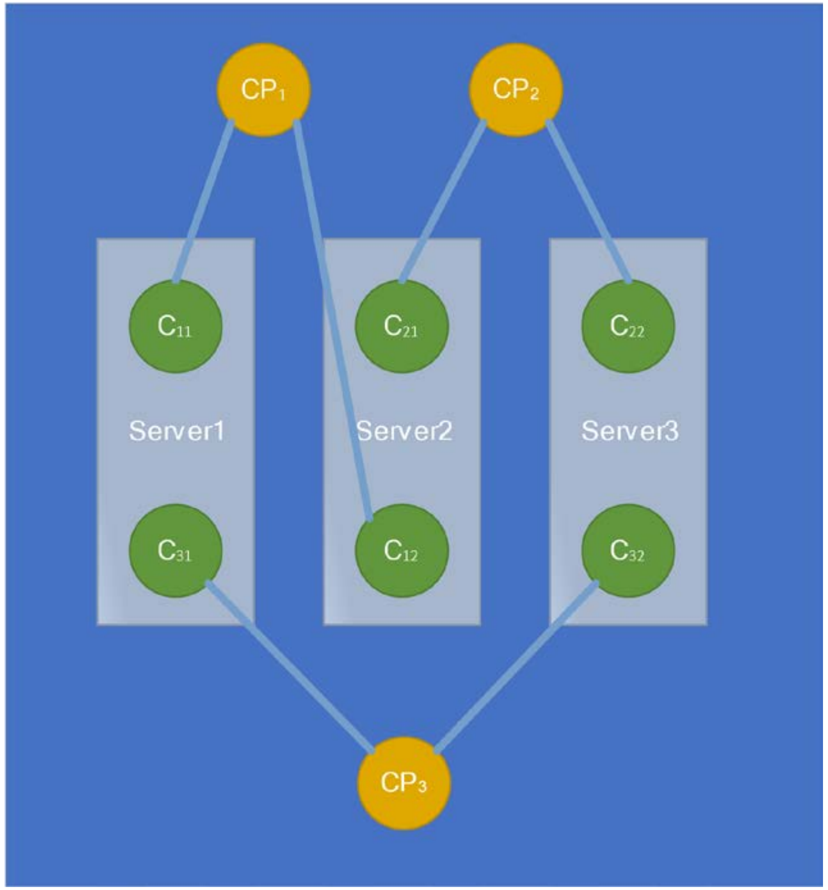
High Availability Model

- **Device handlers are arranged in active/active pairs.**
- **Each pair is considered a virtual device handler.**



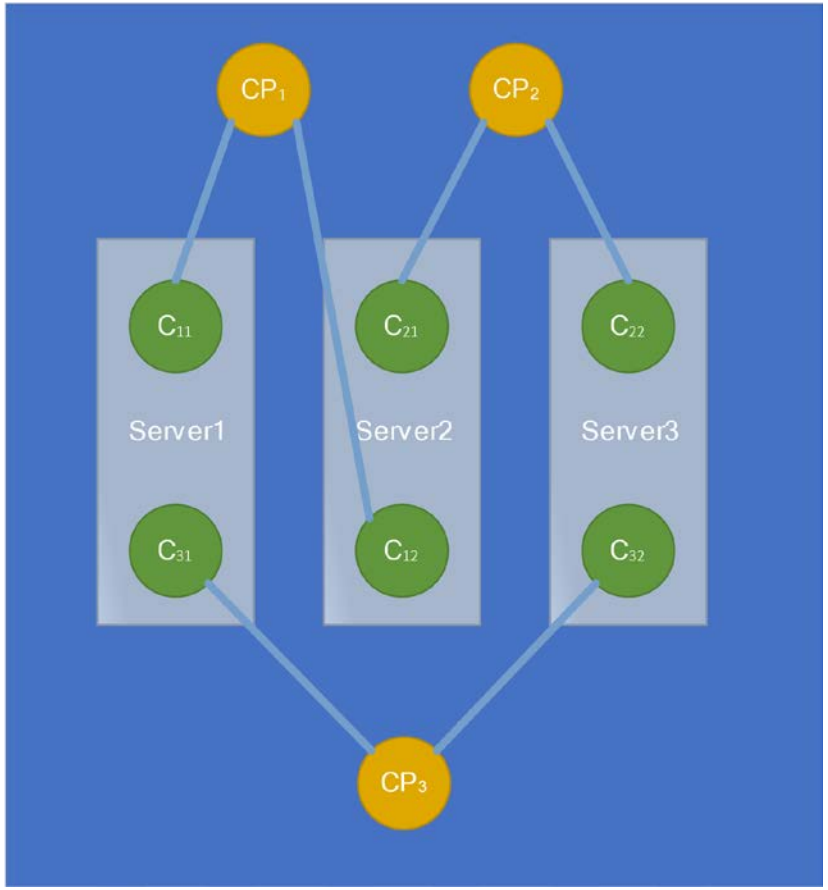
High Availability Model

- **Device handlers are arranged in active/active pairs.**
- **Each pair is considered a virtual device handler.**
- **Affinity is assigned to the virtual device handler / device handler pair.**



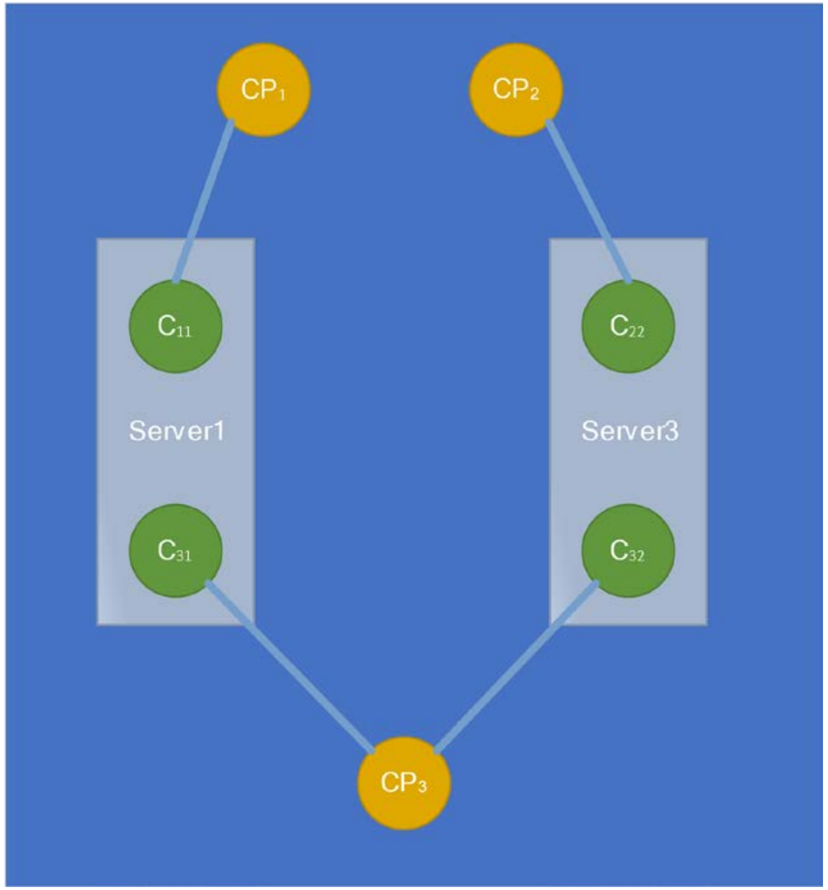
High Availability Model

- **Device handlers are arranged in active/active pairs.**
- **Each pair is considered a virtual device handler.**
- **Affinity is assigned to the virtual device handler / device handler pair.**
- **Round-robin is also handled at the virtual device handler / device hander pair.**



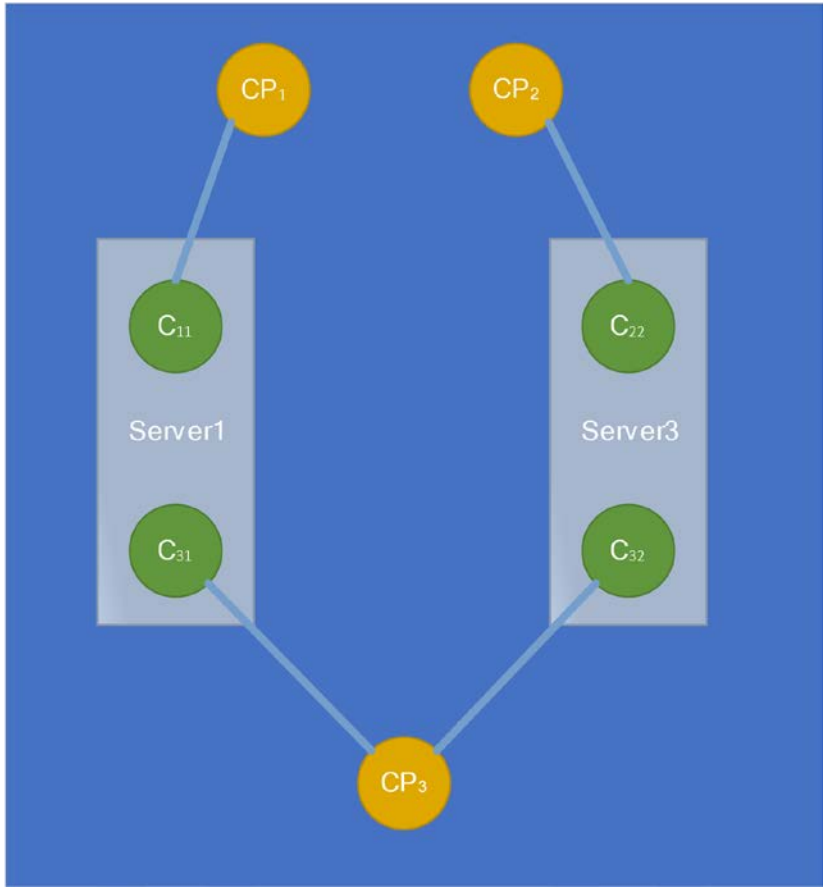
High Availability Model

- **Device handlers are arranged in active/active pairs.**
- **Each pair is considered a virtual device handler.**
- **Affinity is assigned to the virtual device handler / device handler pair.**
- **Round-robin is also handled at the virtual device handler / device hander pair.**
- **The loss of any one device handler or an entire server will not result in any noticeable outage.**



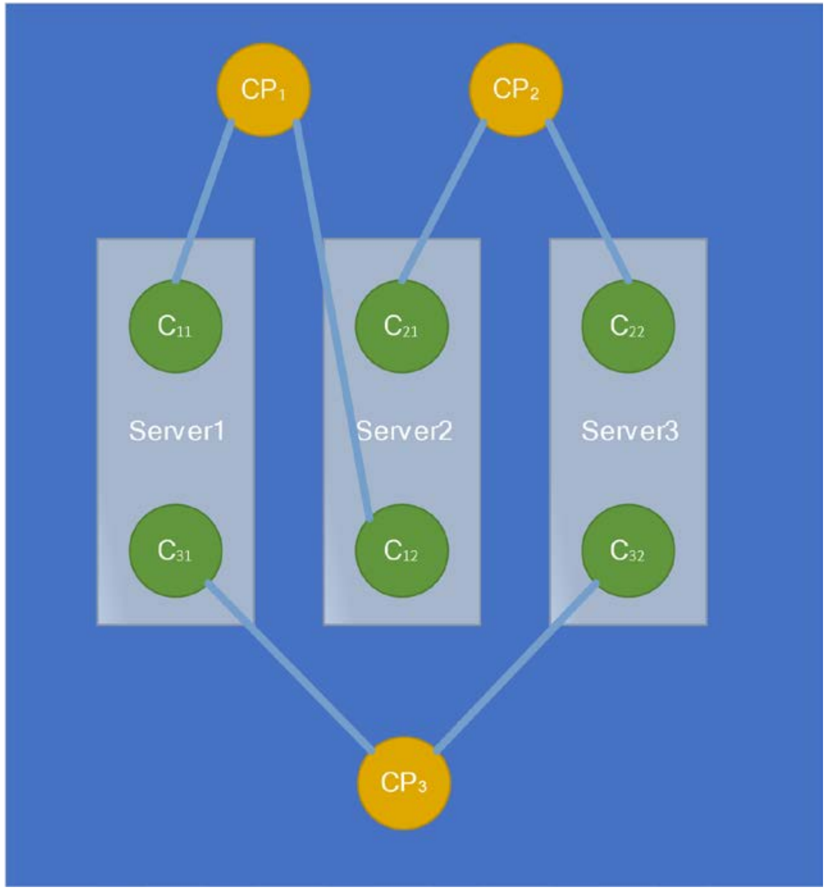
High Availability Model

- Device handlers are arranged in active/active pairs.
- Each pair is considered a virtual device handler.
- Affinity is assigned to the virtual device handler / device handler pair.
- Round-robin is also handled at the virtual device handler / device hander pair.
- The loss of any one device handler or an entire server will not result in any noticeable outage.
- The entire system will continue to function in a high risk configuration

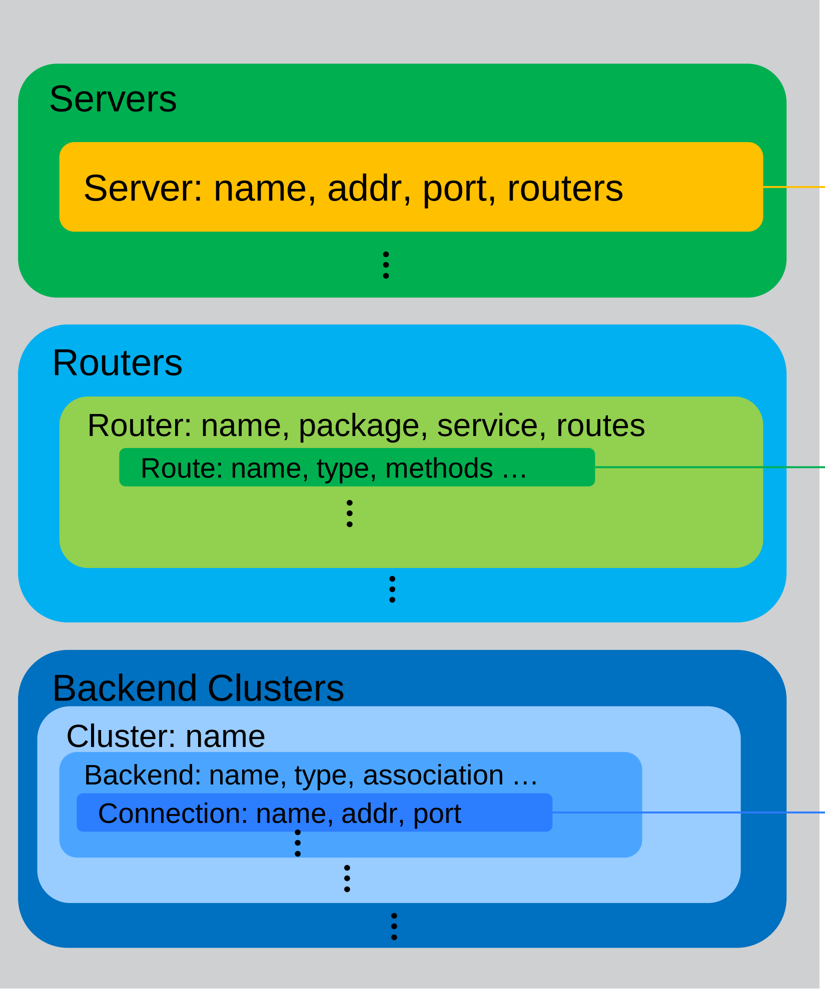


High Availability Model

- Device handlers are arranged in active/active pairs.
- Each pair is considered a virtual device handler.
- Affinity is assigned to the virtual device handler / device handler pair.
- Round-robin is also handled at the virtual device handler / device hander pair.
- The loss of any one device handler or an entire server will not result in any noticeable outage.
- The entire system will continue to function in a high-risk configuration
- Once the server (or pod) is restored, it/they are re-paired with the singletons to re-establish a low-risk configuration.



High Availability Model: Affinity Router Structure



Servers

Server: name, addr, port, routers

⋮

Routers

Router: name, package, service, routes

Route: name, type, methods …

⋮

⋮

Backend Clusters

Cluster: name

Backend: name, type, association …

Connection: name, addr, port

⋮

⋮

⋮

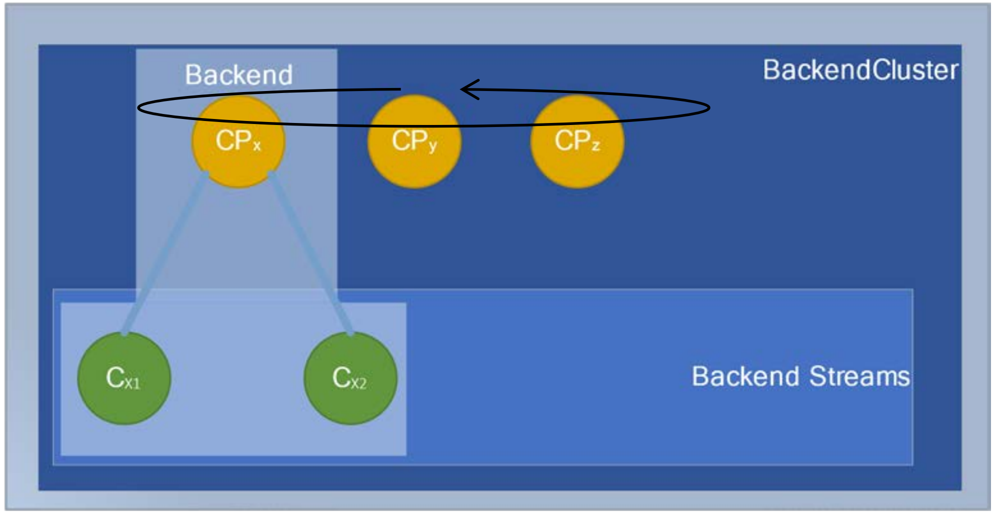
- A server may reference multiple routers
- Routers are keyed by proto package & service
- Only one router per package/service can be defined per server

- Multiple routes exist per router.
- Route selection is based on gRPC method.
- Route types include affinity, round robin, and binding.
- Affinity routes use a proto message value as a backend selector key. Initial binding direction (north or south) is selectable by RPC.

- Each cluster can have multiple backends the routing strategy determines which in the backend is used when
- Each backed can have multiple connections. The association determines how identical messages can be identified

High Availability Model

- **Round robin selection occurs at the Backend Cluster**

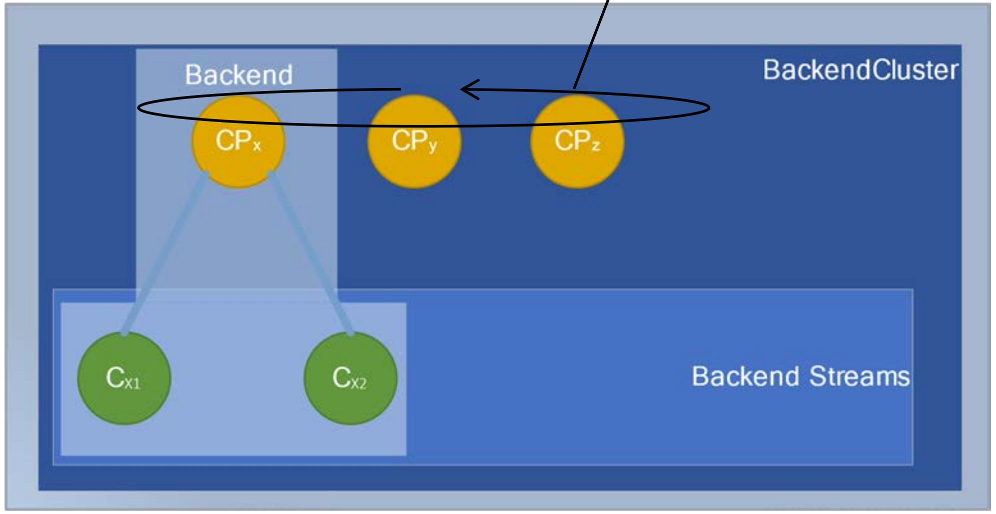


High Availability Model

- **Round robin selection occurs at the Backend Cluster**
- **Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).**
- **CreateDevice binds northbound because the deviceId isn't known until after command execution.**

Package	Service
voltha	VolthaService

DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz

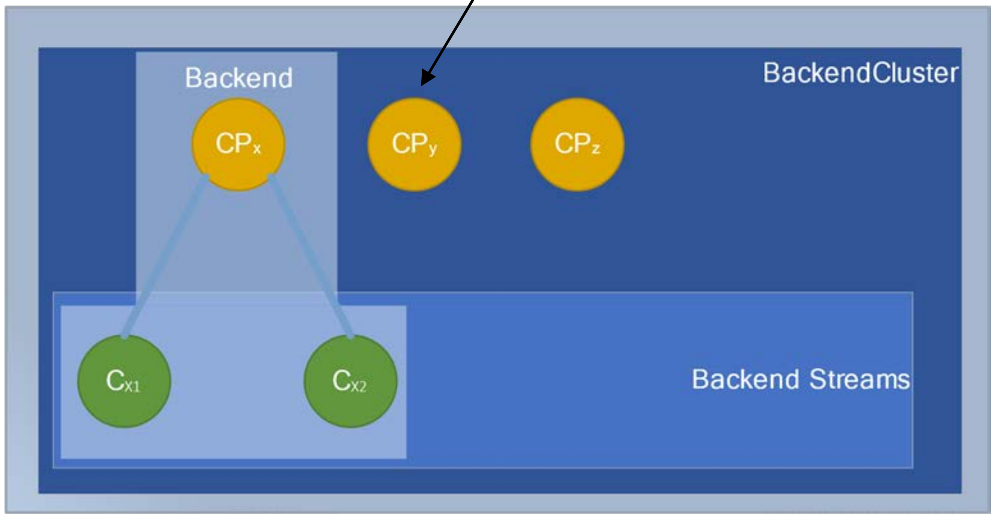


High Availability Model

- **Round robin selection occurs at the Backend Cluster**
- **Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).**
- **CreateDevice binds northbound because the deviceId isn't known until after command execution.**
- **Backend selection of bound devices is made based on the protobuf package, service, and deviceId within the protobuf.**

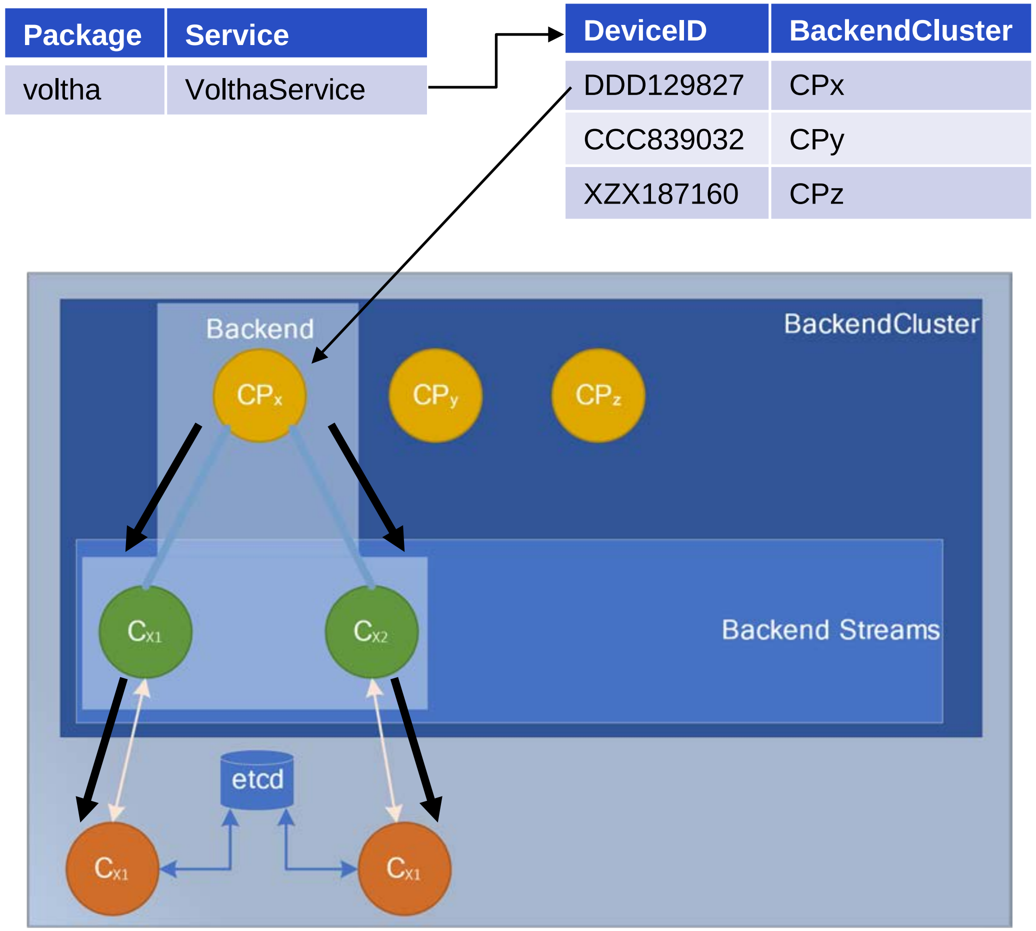
Package	Service
voltha	VolthaService

DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz



High Availability Model

- **Round robin selection occurs at the Backend Cluster**
- **Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).**
- **CreateDevice binds northbound because the deviceId isn't known until after command execution.**
- **Backend selection of bound devices is made based on the protobuf package, service, and deviceId within the protobuf**
- **Requests are sent out both streams to both device handlers with identical serial numbers.**



Package	Service
voltha	VolthaService

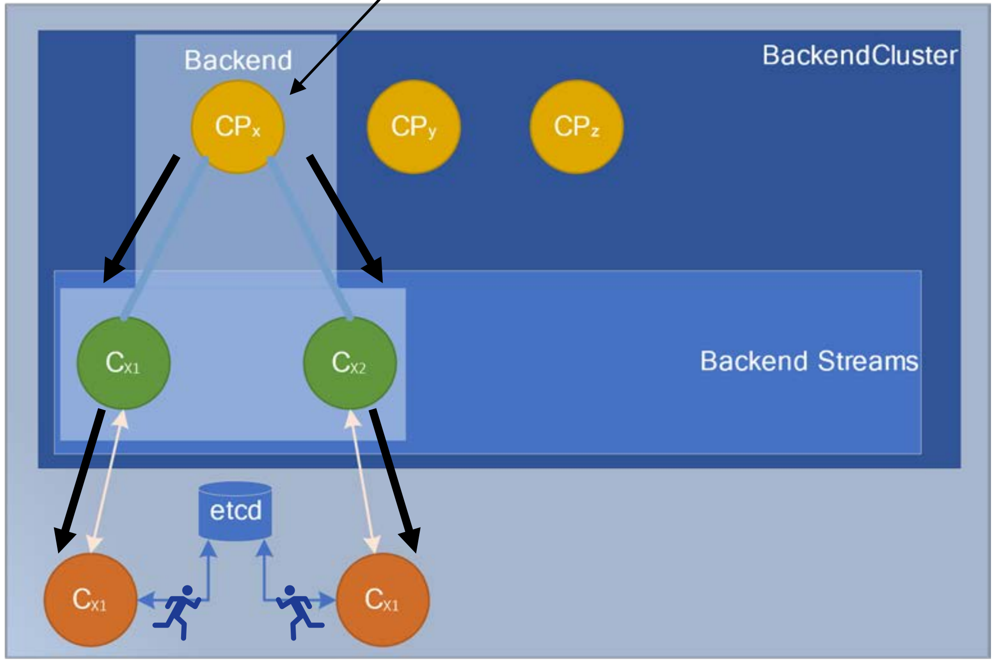
DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz

High Availability Model

- **Round robin selection occurs at the Backend Cluster**
- **Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).**
- **CreateDevice binds northbound because the deviceId isn't known until after command execution.**
- **Backend selection of bound devices is made based on the protobuf package, service, and deviceId within the protobuf**
- **Requests are sent out both streams to both device handlers with identical serial numbers.**
- **Device handlers race to lock a key using the serial number in the KV store.**

Package	Service
voltha	VolthaService

DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz

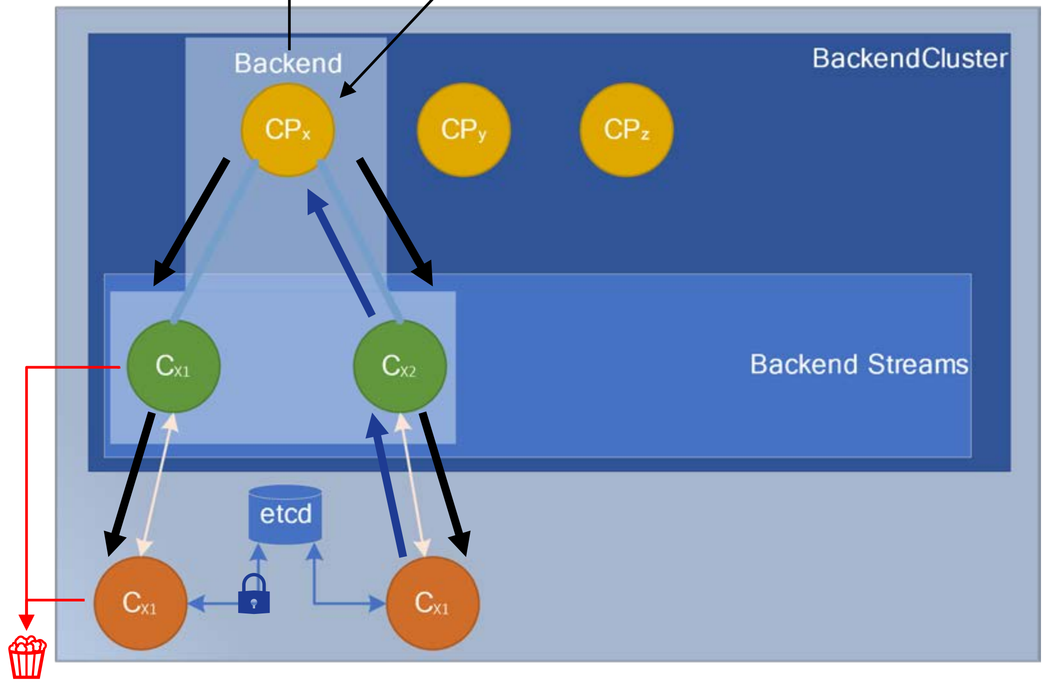


High Availability Model

- Round robin selection occurs at the Backend Cluster
- Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).
- CreateDevice binds northbound because the deviceId isn’t known until after command execution.
- Backend selection of bound devices is made based on the protobuf package, service, and deviceId within the protobuf
- Requests are sent out both streams to both device handlers with identical serial numbers.
- Device handlers race to lock a key using the serial number in the KV store.
- The winner locks out the loser and responds to the request.

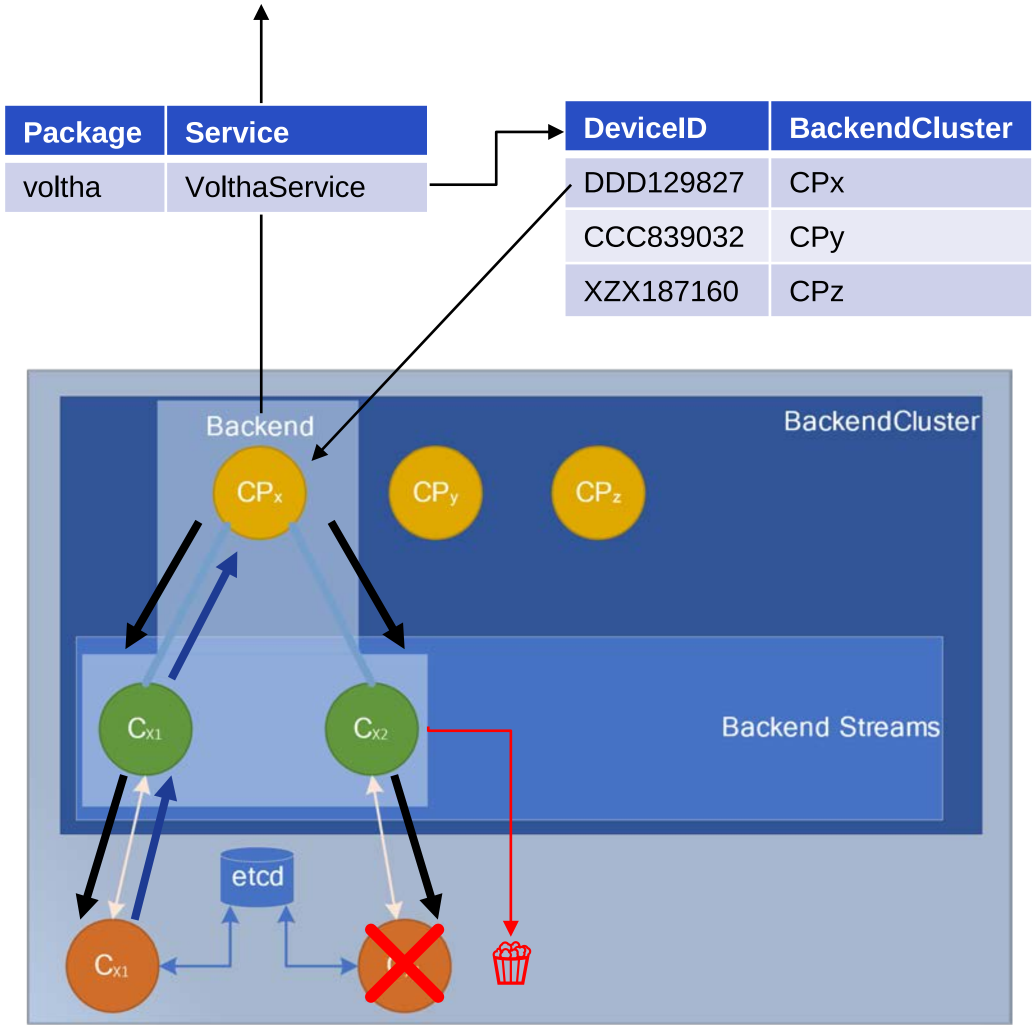
Package	Service
voltha	VolthaService

DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz



High Availability Model

- **Round robin selection occurs at the Backend Cluster**
- **Device to backend affinity binding occurs southbound for all requests except CreateDevice (AKA pre-provision).**
- **CreateDevice binds northbound because the deviceId isn't known until after command execution.**
- **Backend selection of bound devices is made based on the protobuf package, service, and deviceId within the protobuf**
- **Requests are sent out both streams to both device handlers with identical serial numbers.**
- **Device handlers race to lock a key using the serial number in the KV store.**
- **The winner locks out the loser and responds to the request.**
- **The loser waits and should the winner not respond it takes over and provides a response.**



Package	Service
voltha	VolthaService

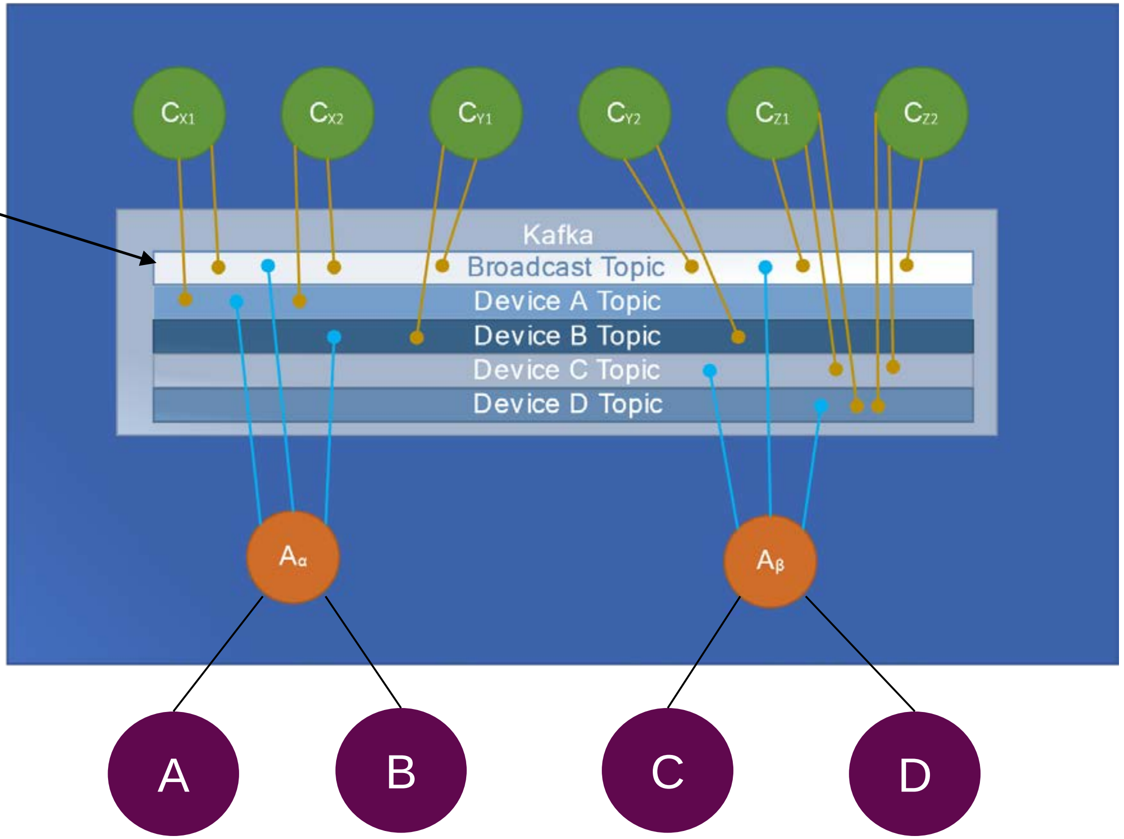
DeviceID	BackendCluster
DDD129827	CPx
CCC839032	CPy
XZX187160	CPz

Subhead Information

Kafka Adapter Messaging Model

Kafka Adapter Message Model

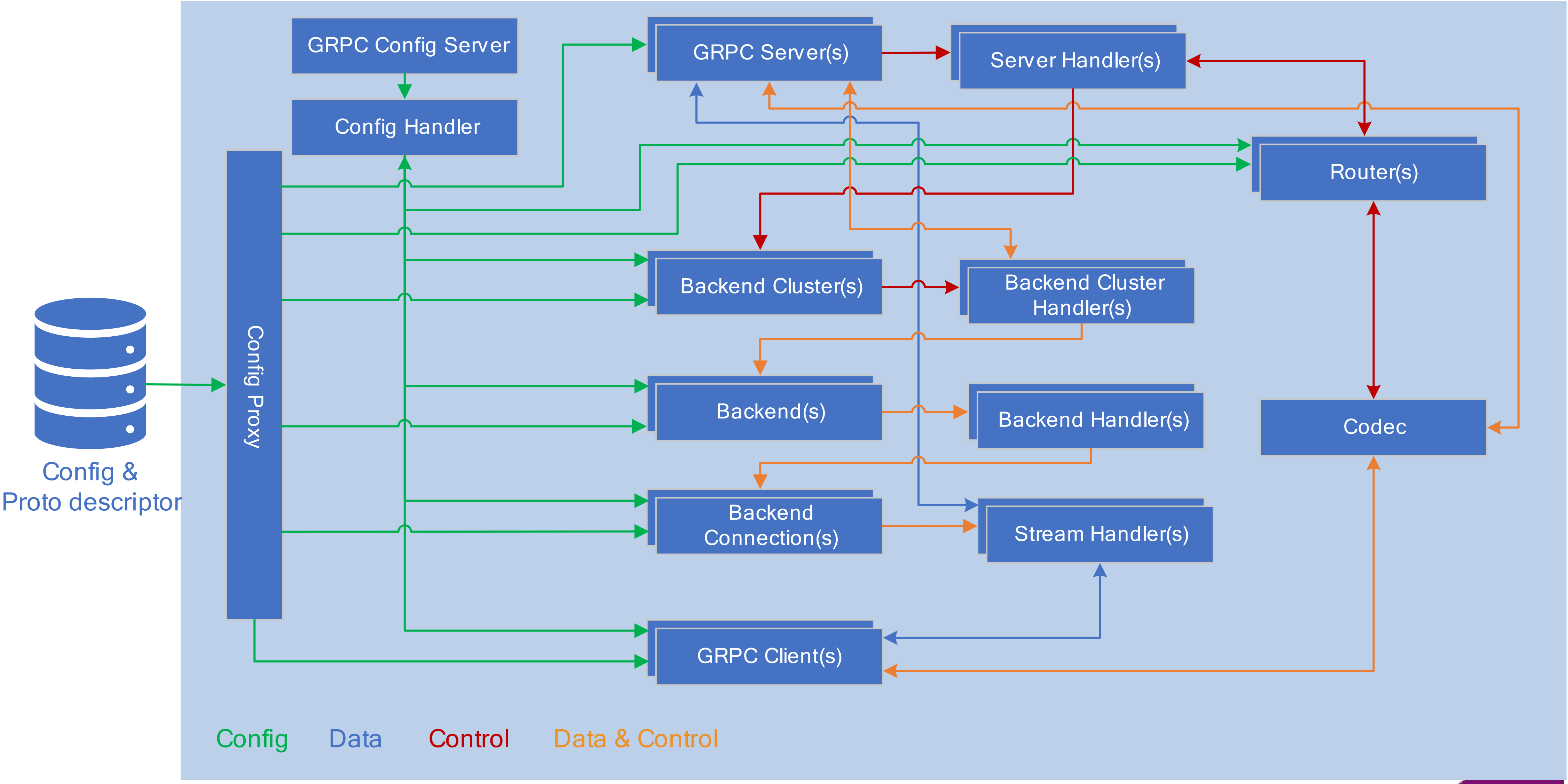
- **A topic is created for every device**
- **A device handler will listen and post on the topics for devices it's handling.**
- **An adapter will listen and post on topics for devices it's managing.**
- **A broadcast topic is used primarily for discovery.**
  - If an adapter can't find a topic for a device it will broadcast it's message on the broadcast topic.
  - One of the device managers will pick up that broadcast.
- **The device manager does the same southbound.**
  - If a topic doesn't exist it will create it.
  - It will broadcast the message on the broadcast topic
  - The corresponding adapter will respond on the newly created topic.



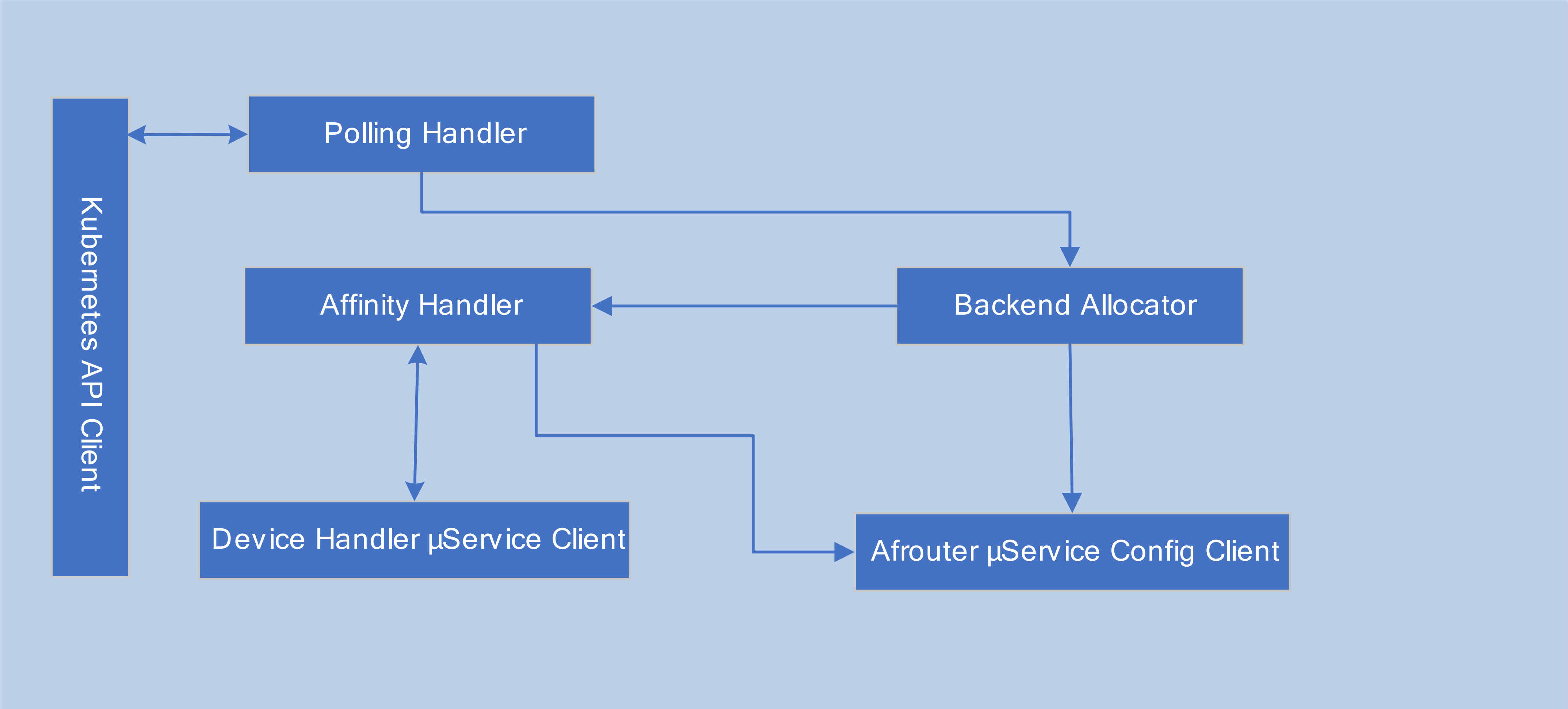
Subhead Information

Per µService Architecture

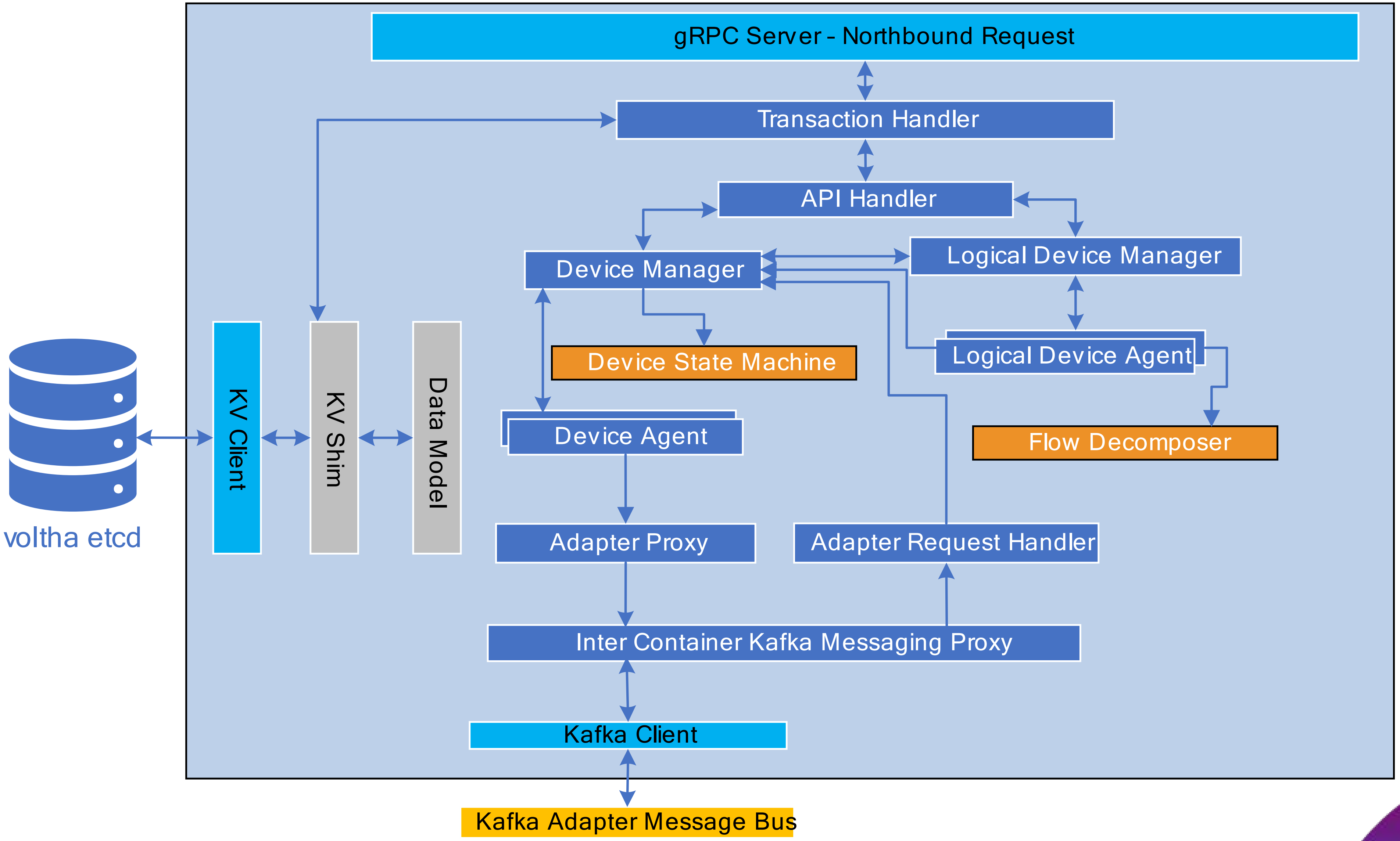
Affinity Router µService



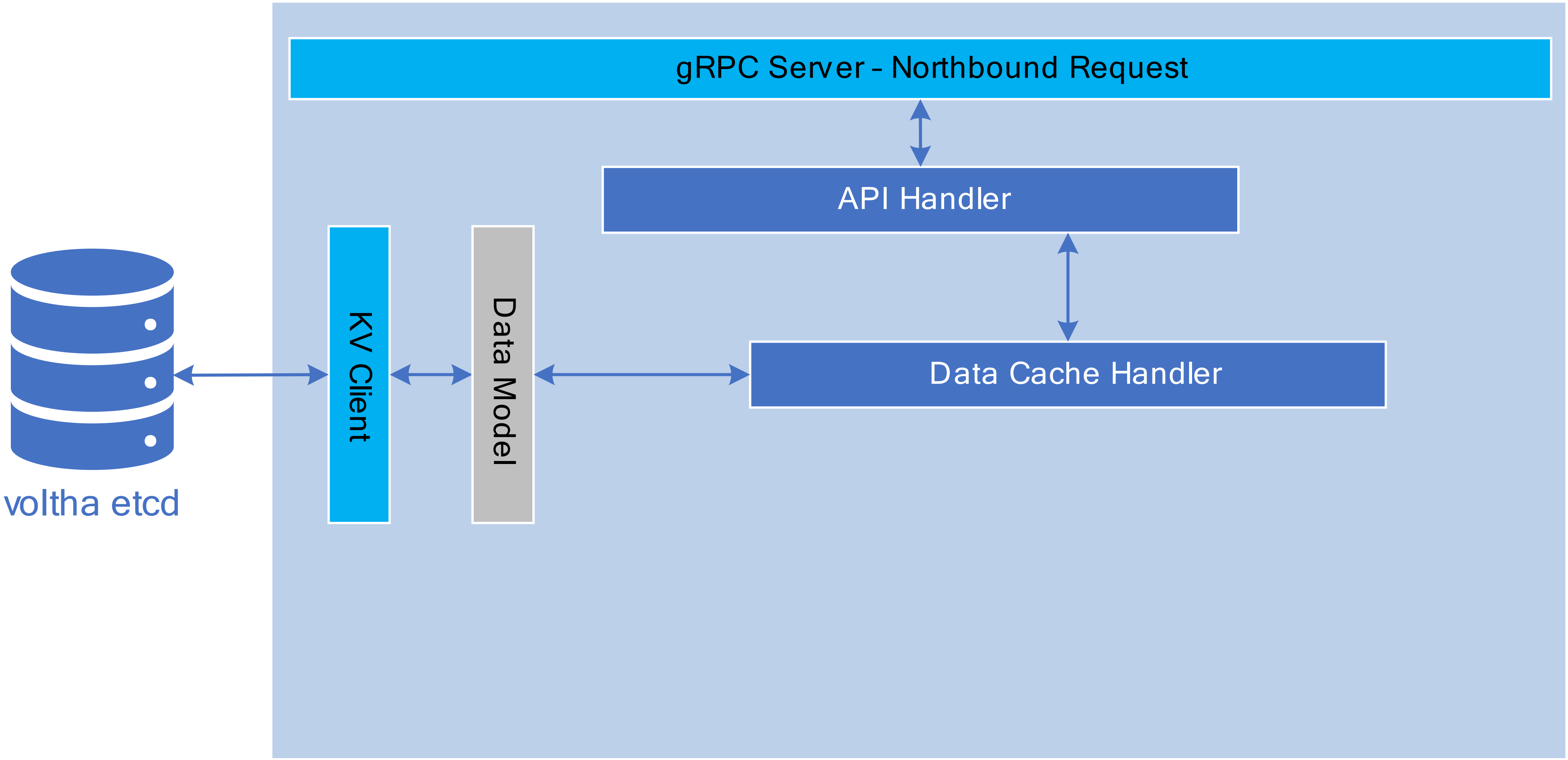
Affinity Router Daemon µService



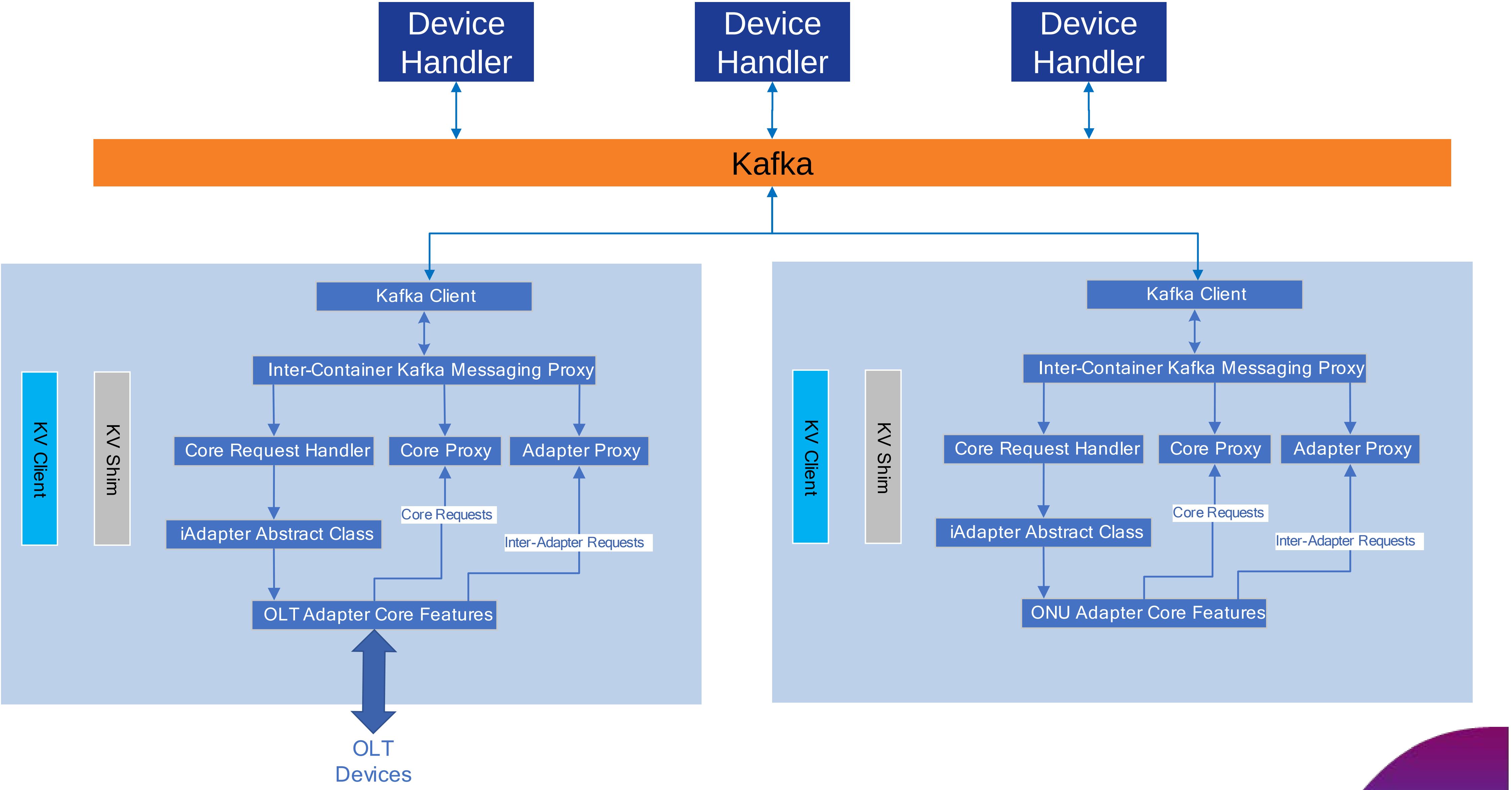
Device Handler µService



Read Only µService



Adapter Shims

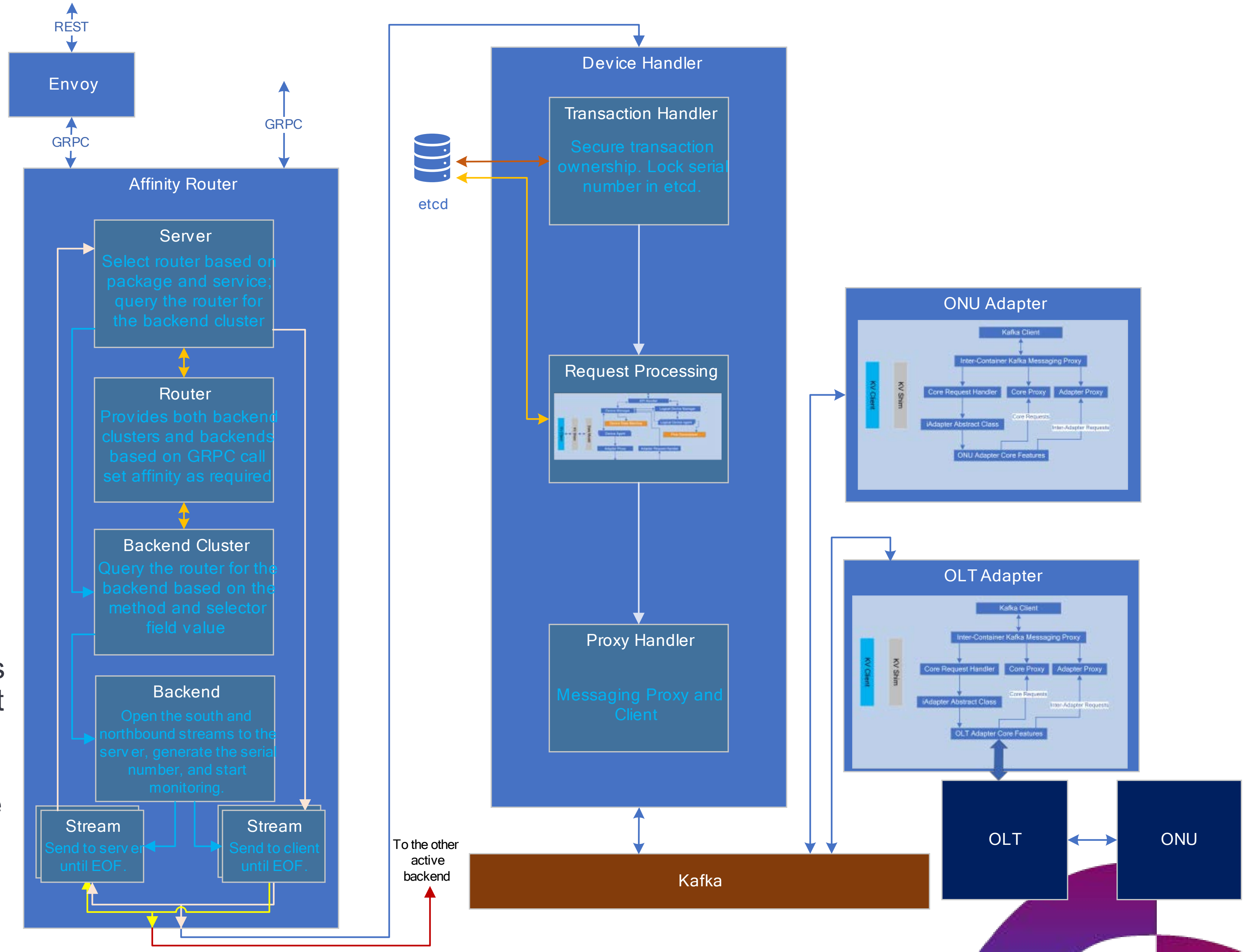


Subhead Information

Transaction Flows Through The System

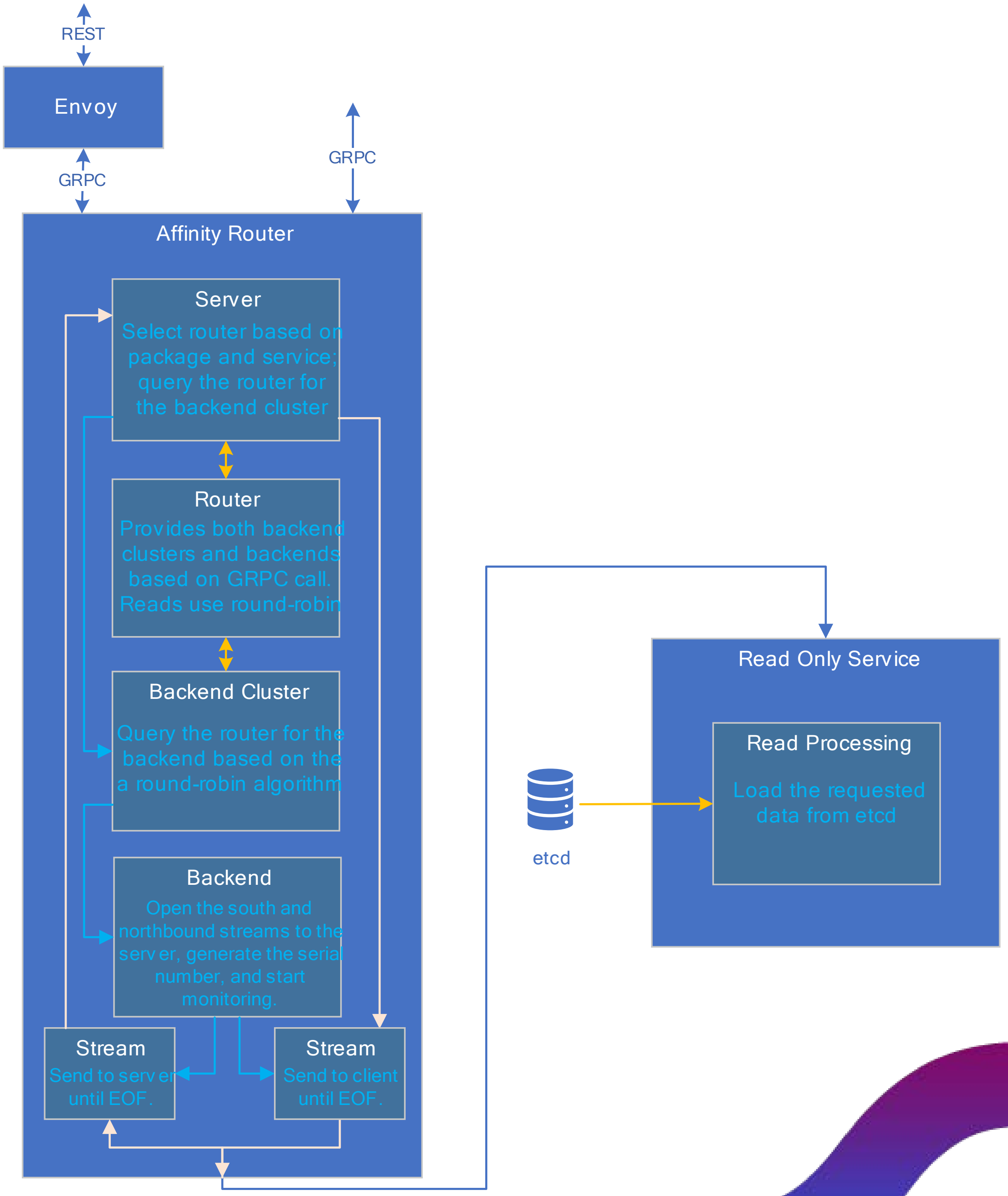
Modify Request

- **Request from NB either REST or GRPC from NB apps.**
- **Affinity routing does one of 2 things (in addition to assigning a serial number for the request)**
  - For pre-provision AKA CreateDevice, request is round-robined to the next core pair. Affinity is established northbound
  - For all other requests existing affinity is used. If no affinity, round-robin to next core pair and establish affinity.
- **The selected core pair does the following:**
  - The first to receive the request locks the serial number in etcd locking out the other pair member.
  - Should the first request not complete the second member of the pair will process the request.


REST
Envoy
GRPC
GRPC
Affinity Router
Server
Select router based on package and service; query the router for the backend cluster
Router
Provides both backend clusters and backends based on GRPC call set affinity as required
Backend Cluster
Query the router for the backend based on the method and selector field value
Backend
Open the south and northbound streams to the server, generate the serial number, and start monitoring.
Stream
Send to server until EOF.
Stream
Send to client until EOF.
etcd
Device Handler
Transaction Handler
Secure transaction ownership. Lock serial number in etcd.
Request Processing
Proxy Handler
Messaging Proxy and Client
To the other active backend
Kafka
ONU Adapter
Kafka Client
Inter-Container Kafka Messaging Proxy
KV Client
KV Shim
Core Request Handler
Core Proxy
Adapter Proxy
iAdapter Abstract Class
Core Requests
Inter-Adapter Requests
ONU Adapter Core Features
OLT Adapter
Kafka Client
Inter-Container Kafka Messaging Proxy
KV Client
KV Shim
Core Request Handler
Core Proxy
Adapter Proxy
iAdapter Abstract Class
Core Requests
Inter-Adapter Requests
OLT Adapter Core Features
OLT
ONU

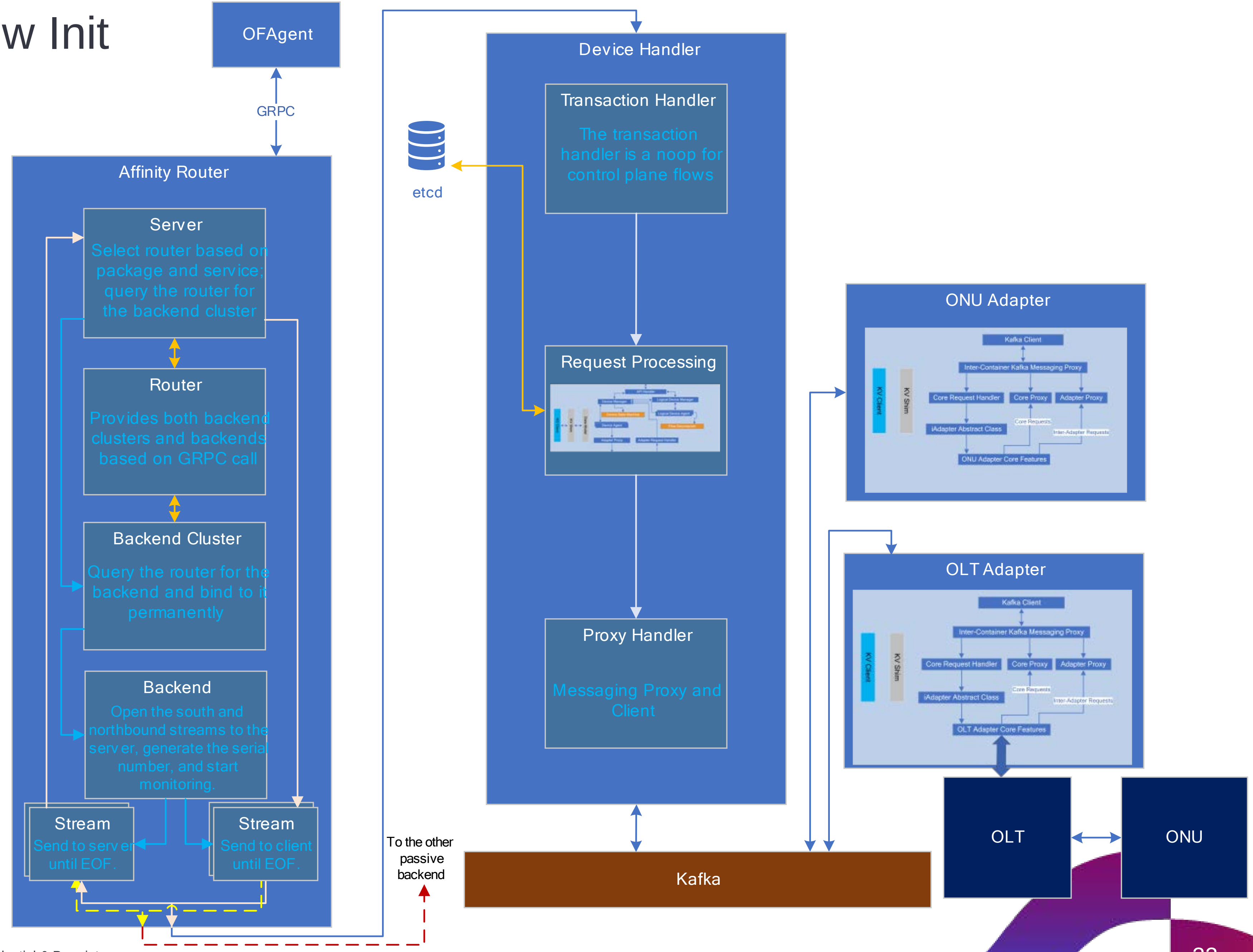
Read Request

- **Request from NB either REST or GRPC from NB apps.**
- **Simplest of all requests.**
- **A round-robin selection is made to one of the R/O cores.**
- **The request is made to that core.**
- **The core reads the requested information from the etcd KV store.**
- **The core uses a caching algorithm to discard older un-used cache entries.**


REST
Envoy
GRPC
GRPC
Affinity Router
Server
Select router based on package and service; query the router for the backend cluster
Router
Provides both backend clusters and backends based on GRPC call. Reads use round-robin
Backend Cluster
Query the router for the backend based on the a round-robin algorithm
Backend
Open the south and northbound streams to the server, generate the serial number, and start monitoring.
Stream
Send to server until EOF.
Stream
Send to client until EOF.
Read Only Service
Read Processing
Load the requested data from etcd
etcd

Control Plane Packet Flow Init

- **The OFAgent initiates a connection through GRPC.**
- **The affinity router uses round-round robin to secure the next backend cluster.**
- **One of the pair is chosen at random to which the communication is bound.**
- **A stream is created that persists until**
  - Someone closes it.
  - The chosen pair member disconnects.
- **In the case of a disconnect (not EOF). The stream is immediately switched to the alternate pair member.**



OFAgent

GRPC

Affinity Router

Server

Select router based on package and service; query the router for the backend cluster

Router

Provides both backend clusters and backends based on GRPC call

Backend Cluster

Query the router for the backend and bind to it permanently

Backend

Open the south and northbound streams to the serv er, generate the serial number, and start monitoring.

Stream

Send to serv er until EOF.

Stream

Send to client until EOF.

To the other passive backend

etcd

Device Handler

Transaction Handler

The transaction handler is a noop for control plane flows

Request Processing

Proxy Handler

Messaging Proxy and Client

Kafka

ONU Adapter

OLT Adapter

OLT

ONU

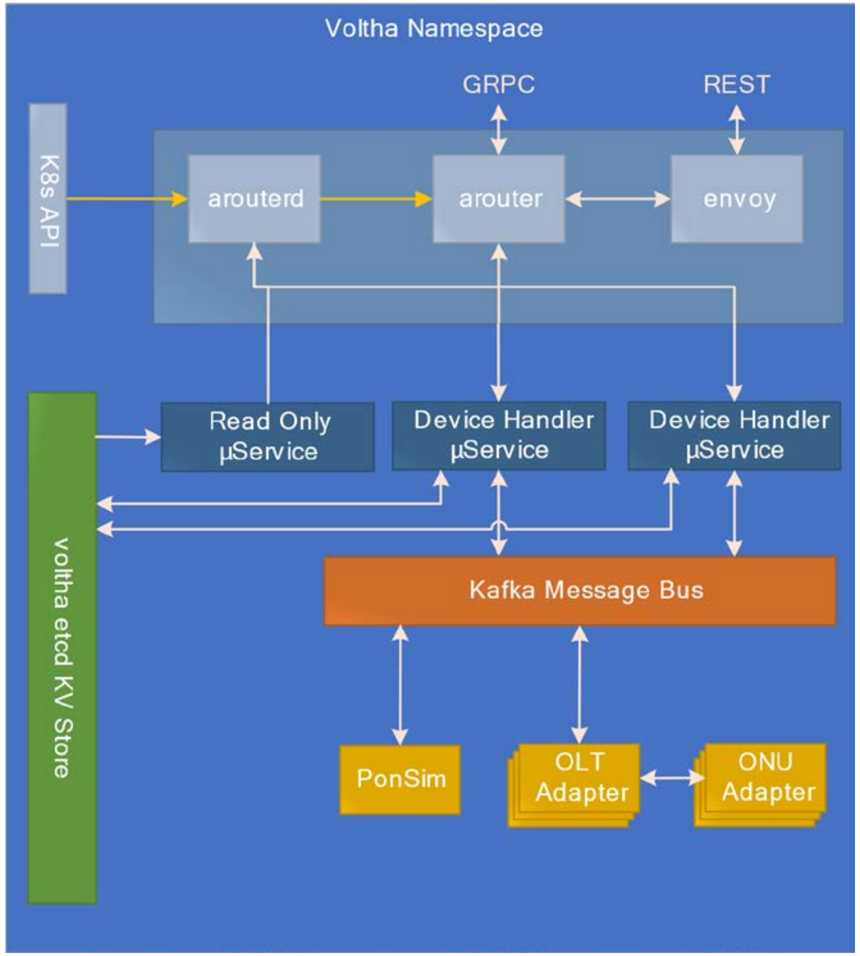
Subhead Information

Kubernetes Integration

ciena

K8s Integration

- **Each µService is run in its own pod with one exception**
- **The affinity routing proxy pod hosts 3 µServices**
  - The primary service is the arouter service
  - Two sidecar services (envoy & arouterd)
  - envoy is used to map GRPC ←→ REST
  - arouterd configures the arouter and device handlers depending on the context.
- **At afrouter pod startup the following happens**
  - arouterd queries k8s for all pods
  - Each device handler is queried for devices
  - An intersection algorithm is used to pair the handlers and the config is pushed to the arouter.
- **During normal operations**
  - arouterd queries k8s for all pods and maintains last state.
  - If pod state changes arouter config is pushed to reflect current status.
    - If a device handler pod is lost its backend is removed from the config
    - If a device handler pod returns then it’s provided a list of ID’s it should have and its backend is added back to the config.



Thank You

ciena®

Experience. Outcomes.