KubeCon | CloudNativeCon

North America 2017

Building an Edge Computing Platform for Network Services Using Cloud Native Technologies

Stephen Wong, *FutureWei Technologies, Inc.*

Agenda

1. Problem We Looked at
2. Framework That We Came Up with
3. Demo
4. Future Roadmap

NFV in the Context of Cloud Native Computing

- Who are we?
    - OPNFV contributors, focusing on examining cloud native computing's relevancy in NFV use cases
- NFV —- VNF
    - High-level definition of NFV: idea that majority of network functions could be provided by software running on top of COTS servers
    - High-level definition of VNF (virtual network function) —- a piece of software responsible to run a single network function
    - The problem we are looking at: can we build VNFs as cloud native applications? If we can, we get benefits of:
        - Scalability
        - Resiliency / Fault Tolerance
        - Composability and reusability
        - Ease of development and deployment

Edge Network Services Applicability

- Edge use cases are suitable for cloud native VNFs due to:
  1. Resource constraints: therefore micro-service and containerization provide more optimal resource utilizations
  2. High Cost of Edge Maintenance: resiliency and fault tolerant nature of cloud native applications would be beneficial
  3. Ease of Deployment: suitable for remote push of new images of components, allows rollback if new component fails locally
  4. Replicable: entire deployment can be replicated to different sites

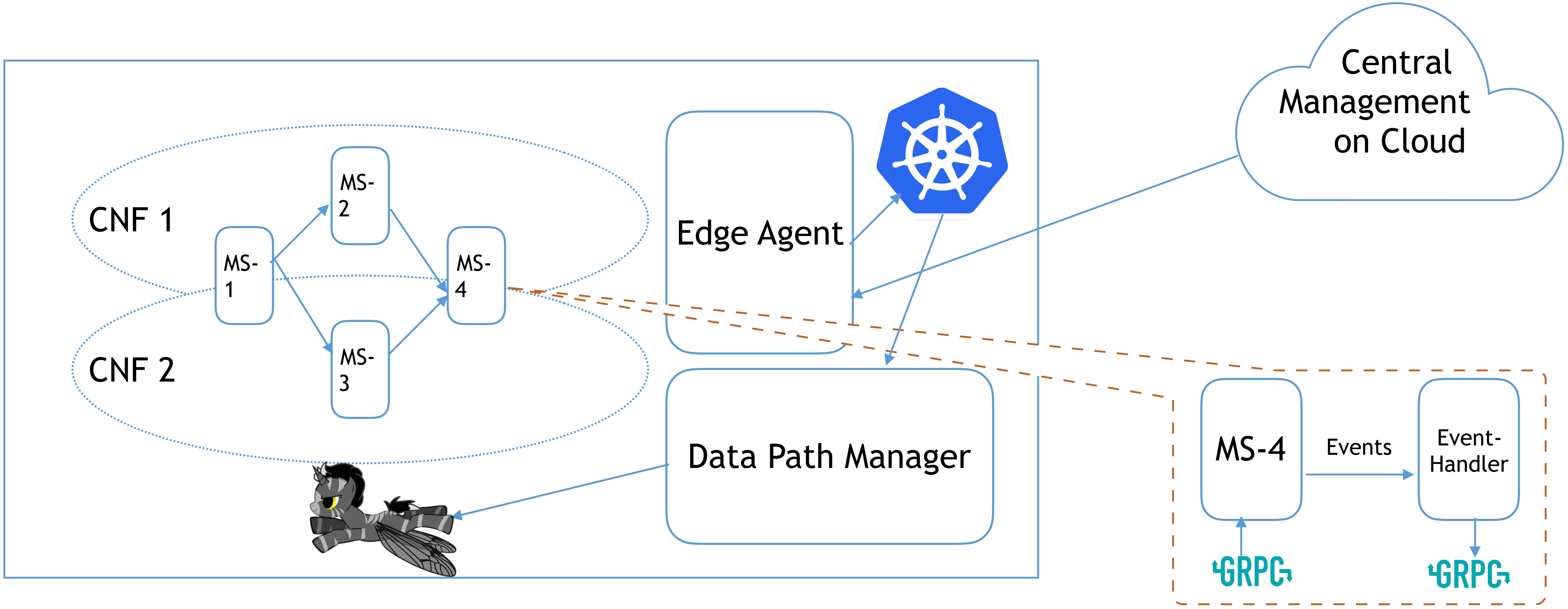
Two Gaps in Building VNF as Cloud Native Application

1. VNF by nature deals with transport traffic, as opposed to requests destined to a service or node in cluster:
   - A subset of these network functions act similarly to a transparent proxy, i.e., something which does not expose endpoint to users, but system needs to route traffic to
   - This is typically referred to as “service insertion”
   - At least two modes of operations: redirect traffic (traffic now routed to service) or tap mode (traffic mirrored to service), and good to provide programmability from the individual microservicres: deny, or QoS (rate limiting)
   - Bring Your Own Datapath
2. VNFFG (VNF Forwarding Graph)
   - An orchestration function where operator can specify the chain of VNFs user traffic would traverse
   - The realization of an FG is typically a data path function
   - For cloud native VNF, this forward graph basically becomes a service graph in a service mesh
     - The main difference being, that the service graph isn’t built into the application, but is operator defined

Terminology

1. CNF: **C**loud-native **N**etwork **F**unction —- defined by *operator*, made up of a list of *μsrvs* (*microservices)*
2. Client: initiates traffic which would pass through the edge node
3. μsrv (microservice): a unit of compute service which exposes a set of APIs, the “lego pieces” that would make up a CNF
4. Operator: defines and deploys CNFs

Architecture of Our Approach

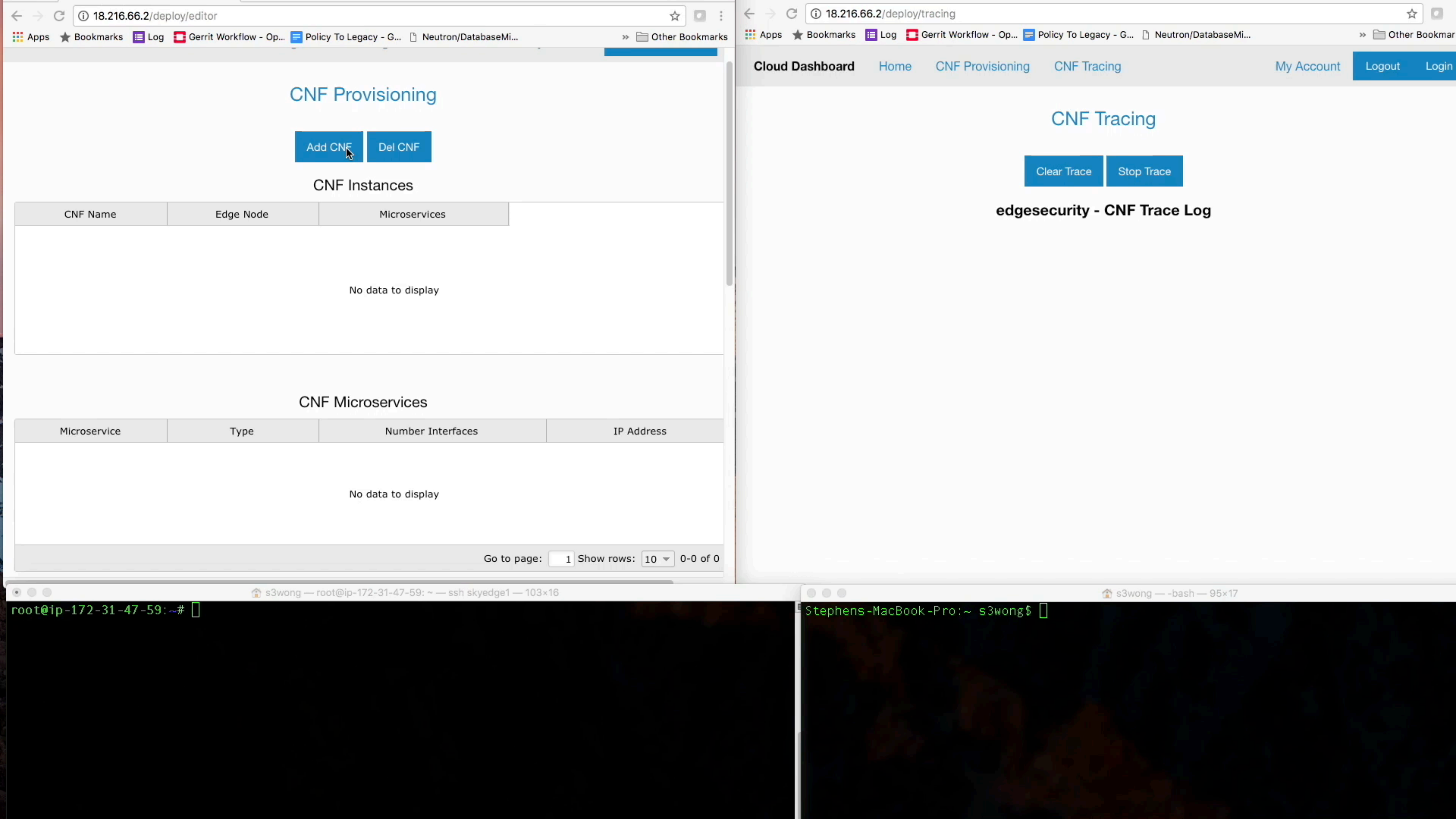


Technical Descriptions

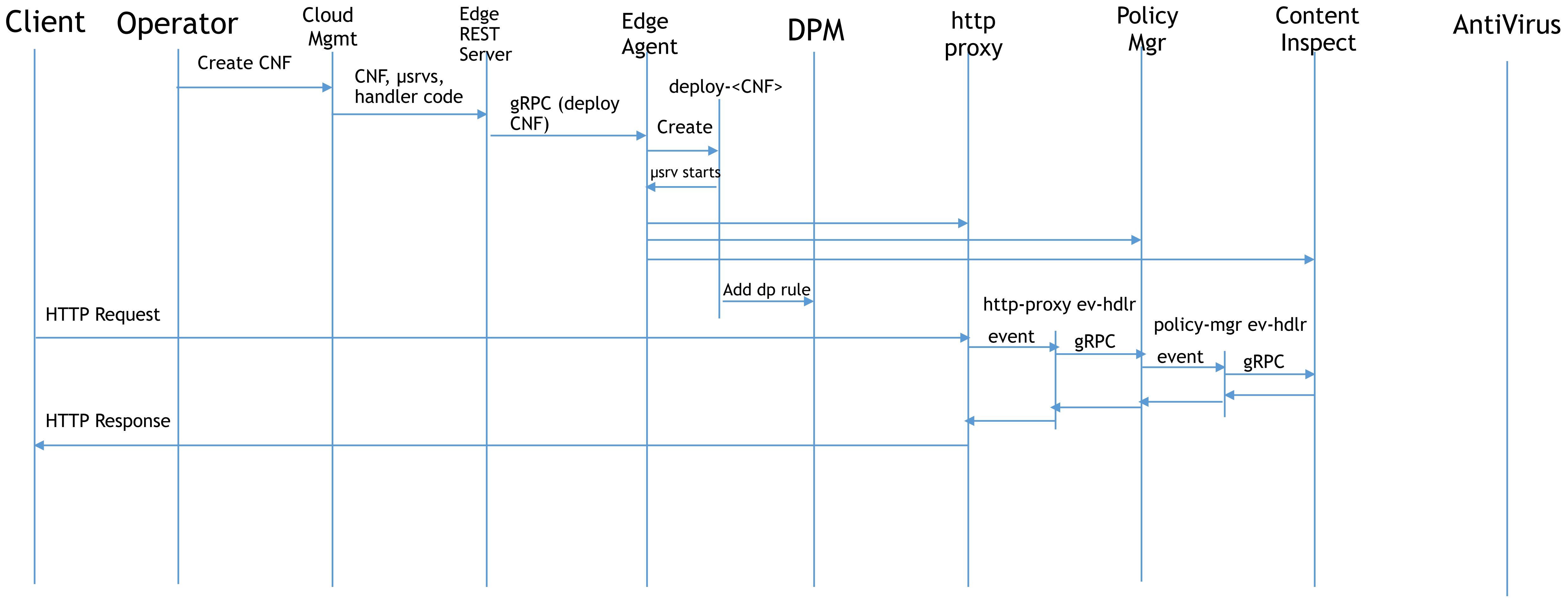
1. Event-driven Model
    1) Event handler running as sidecar in the same pod as μsrvs
    2) μsrvs expose a set of events, which operators implement code to handle events they selected and dynamically build and push that event-handler to co-exist with the μsrv
    3) An SDK would be provided by the framework. Operators invoke SDK calls in their custom event handlers, then internally those SDK calls would map to gRPC / REST API calls
    4) Operators therefore are implementing programs to build service graph
2. Programmable Datapath
    1) Takes network policy rules and program data path engine accordingly
    2) Rule format : { <ip> port : {redirect | copy} => [list of labels]}
    3) Utilizes eBPF (IOVisor bcc) to load BPF code
        - Possibly allowing μsrv to inject its own datapath
    4) Datapath Manager exposes gRPC APIs, any μsrv (via event handler) can invoke gRPC to program datapath

Demo Description

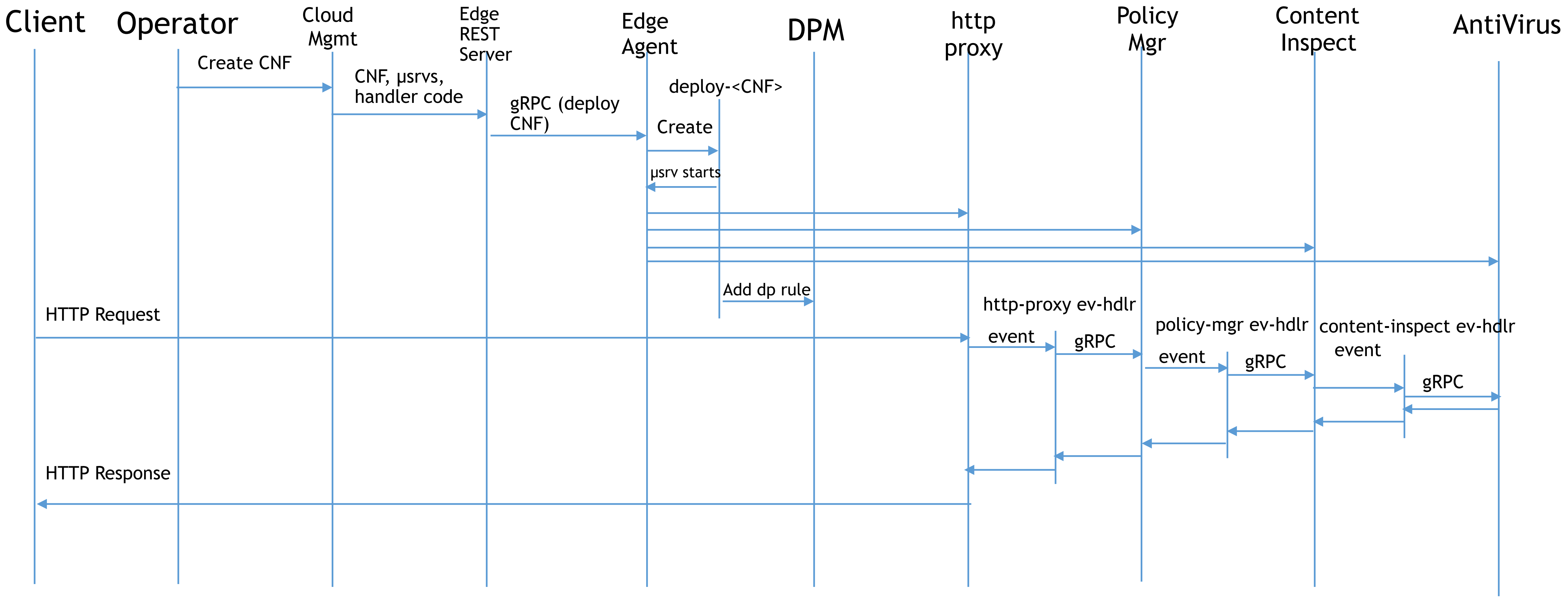
1. Operator uses the portal to define a CNF (edgesecurity —- protecting clients) which includes multiple µsrvs (http-proxy, policy-mgr, content-inspect), each µsrv publishes a set of events, and operator can choose to implement these event-handlers and customized them via a set of Python SDK methods (also specify network policy rules to direct traffic to the http-proxy), and deploy the CNF to a target edge node
2. Previously non running µsrvs (http-proxy, policy-mgr, content-inspect) now running on edge node, with the new event-handler (per µsrv) built/pushed running alongside the core µsrv (running as a container)
3. Client sends an HTTP request —- the transport traffic is being redirected to the specified pod associated with the datapath policy's 'redirect' action's label; and subsequently the call sequence matches what the operator specified
4. Operator updates portal to add a µsrv (anti-virus) with updating a previous SDK call (anti_virus.scan SDK call) from an existing event handler (inspect_done event from content-inspect)
5. After CNF (re)deploy, the call flow now goes to new µsrv (anti-virus) with the correct RPC
6. Operator wants to narrow what type of traffic should get virus scan (as opposed to everything), thus updating the event_handlers (remove anti_virus.scan from content-inspect's inspect_done, then add it as a condition under policy_check from policy-mgr)


18.216.66.2/deploy/editor
Apps
Bookmarks
Log
Gerrit Workflow - Op...
Policy To Legacy - G...
Neutron/DatabaseMi...
Other Bookmarks
CNF Provisioning
Add CNF
Del CNF
CNF Instances
CNF Name
Edge Node
Microservices
No data to display
CNF Microservices
Microservice
Type
Number Interfaces
IP Address
No data to display
Go to page: 1 Show rows: 10 0-0 of 0
18.216.66.2/deploy/tracing
Cloud Dashboard
Home
CNF Provisioning
CNF Tracing
My Account
Logout
Login
CNF Tracing
Clear Trace
Stop Trace
edgesecurity - CNF Trace Log
s3wong — root@ip-172-31-47-59: ~ — ssh skyedge1 — 103×16
root@ip-172-31-47-59:~#
s3wong — -bash — 95×17
Stephens-MacBook-Pro:~ s3wong$

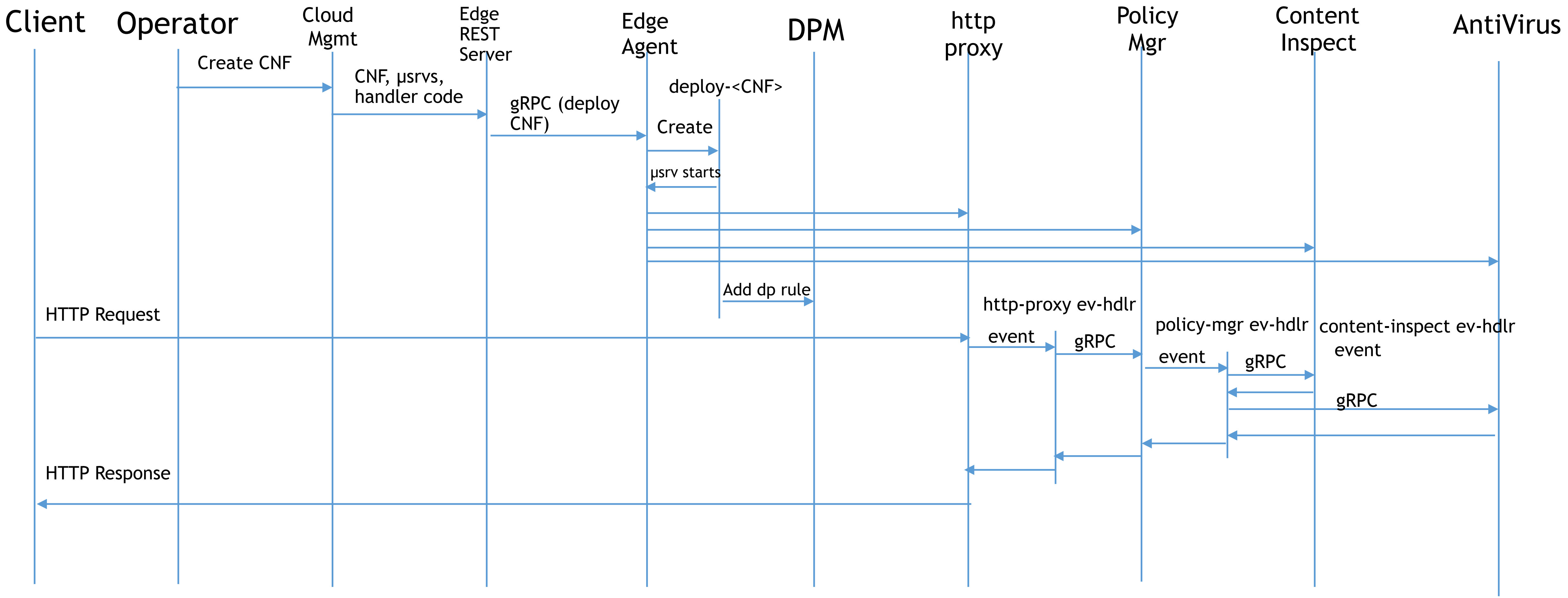
Demo Flow —- Deploy



Demo Flow —- Update Deployment



Demo Flow —- Change AV Invocation Logic



Roadmap (1): Data Collection

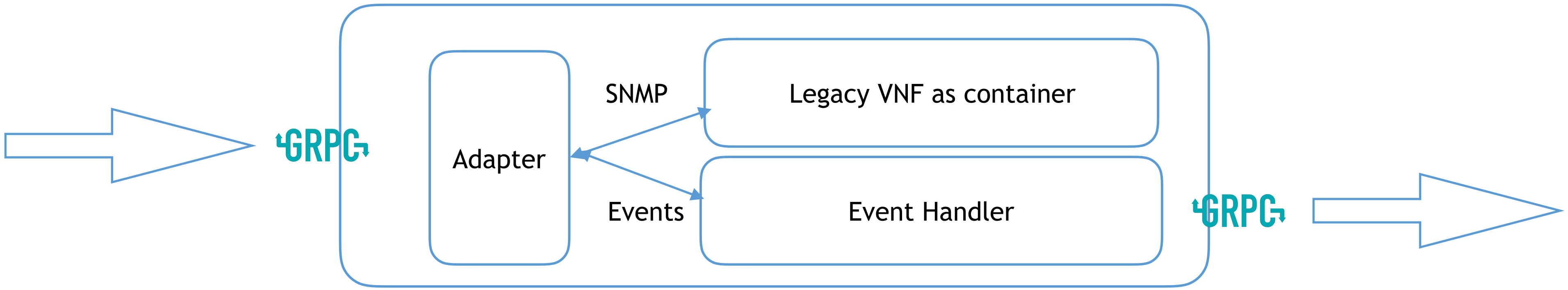
- For edge use cases, it is important to generate rich set of collectable data
- For filtering, allow software-defined filters for what data is being sent to centralized management
- As networking solution, network tracing is important
  - An important reason why eBPF is chosen for datapath engine
  - Particularly IOVisor bcc comes with a rich set of kernel tracing code
- Data correlation: everything throughout the stack (OpenTracing calls, eBPF map, redis objects…etc) all have exact same key (the **cookie** parameter being passed throughout the events / gRPC calls), thus allowing us to bundle related data from various sources

Roadmap (2): Multiple Event Handlers per MicroService

- A major drawback of the current/demo implementation is that the µsrvs are non-sharable, i.e. how the event_handler is invoked per pod makes each µsrv only usable by one CNF
- Moving forward, to allow µsrv to be used by multiple CNFs, cookie filtering value can be used, and event channel can be pub/sub
- Request side needs to make a decision on which CNF a session belongs to (i.e., and most commonly, by client or user id), such info needs to be propagated to **ALL** event_handler on all associated µsrv pods
- Programmable session => CNF correlation?

Roadmap (3): Legacy VNF Integration

- There exists a rich and widely deployed set VNFs used by various operators —- it is naive and irresponsible to believe VNFs needed to be rewritten to fit this cloud native model
- The two requirements for a µsrv in this framework:
    1. Event generation (and synchronous or asynchronous handlers for those events)
    2. gRPC request handling
- The above requirements do assume the VNFs have some degree of programmability
- One solution to incorporate legacy VNF in this framework would be having another container per pod as an adapter to communicate with legacy VNF (as µsrv). The adapter would interface with the legacy VNF (ex: SNMP, CLI) whilst implementing a gRPC server with event generating logic —- in fact, the AV µsrv is implemented this way



Roadmap (4): Cloud Native Computing Ecosystem

- Service Mesh: If scaling up is needed, this framework matches well with the current service mesh architecture (such as Istio)
  - As the framework utilizes gRPC for request to μsrvs, and the SDK calls are implemented as gRPC client calls, this framework naturally fits into HTTP based service mesh
  - Utilizing the “service graph” tool from Istio would greatly increase visibility to operators
- The Tracing shown in demo was quite ad-hoc, but did demonstrate the importance of call tracing in this framework
  - Will examine use of OpenTracing APIs and possibly Jaeger as tracer
- With cloud native VNF, we can leverage existing k8s / cloud native application oriented CI/CD solutions like Spinnaker

Information

- Project Clover under OPNFV (https://wiki.opnfv.org/display/CLOV)
  - Clover intends to investigate the issues and projects associated with building VNFs as cloud native applications
- Slack channel: clover-project
- Email Address: stephen.kf.wong@gmail.com

THANK YOU!!!