#RP55

Kees de Jong
Anas Younis

Sharing digital objects using NDN: PID interoperability, planning and scaling

SeaDataCloud

- SeaDataCloud is a distributed marine data infrastructure network in different geographical domains
  - 8 institutes with over 100 data centers
  - Aiming to make research data available to scientists
- Sharing large data sets becomes a challenge
  - Congestion
  - Interoperability

SeaDataCloud

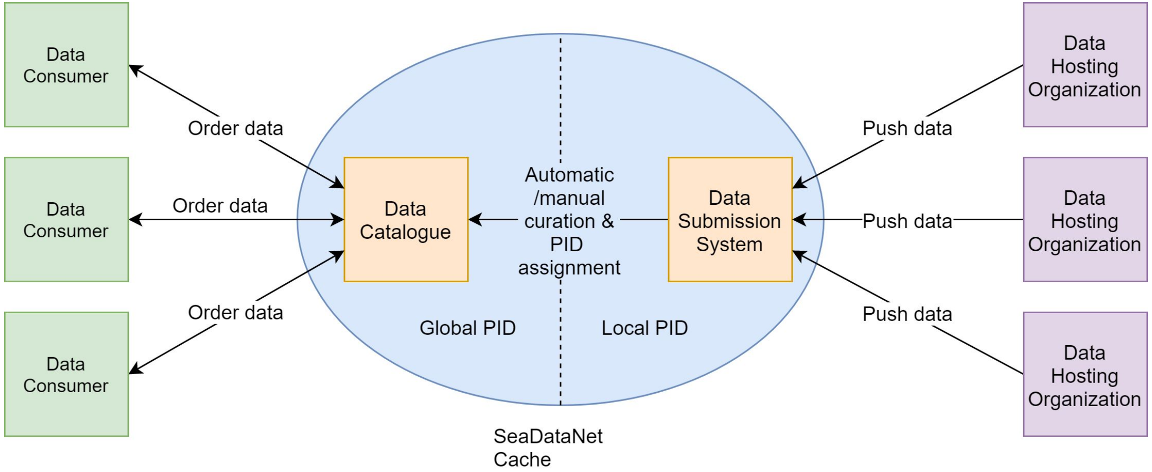


Figure 1: Current SeaData cloud setup

SeaDataCloud

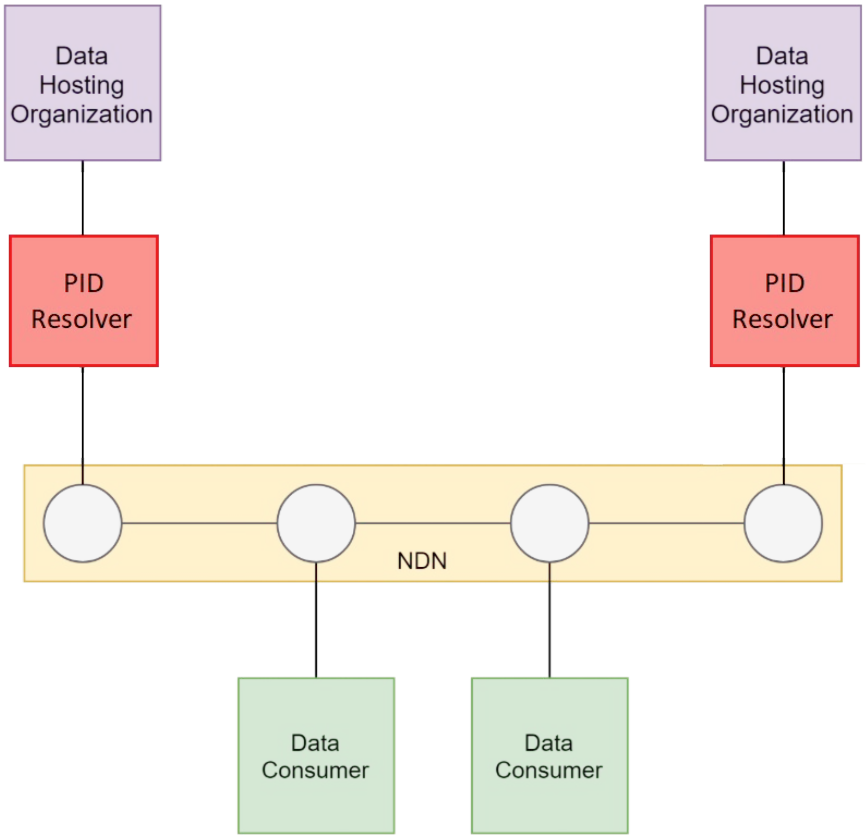


Figure 2: Potential solution

Research question

- How to make the Persistent Identifier (PID) and NDN (Named Data Networking) namespace **interoperable**?
  - How to support different PID types?
  - How to incorporate extensibility for future PID schemes?
- How to **plan** and **scale** an NDN network?
  - Which NDN scaling problems are known?
  - Which method can be used to plan an NDN network?
  - How to deploy an NDN network in a scalable way?

Outline

- Short introduction about NDN and PID
- Related work
- System architecture and virtualized NDN functions
  - PID interoperability
  - Virtual NDN planning, automation and scaling
- Experiment results
- Conclusion and future work

Why NDN?

- NDN is the most mature variation of ICN
  - ICN = Information Centric Networking
  - ndn-cxx solution was used in our proof of concept
- Forwarding based on name prefixes rather than IP
  - No end-to-end connections needed
  - Data cached on intermediary hops

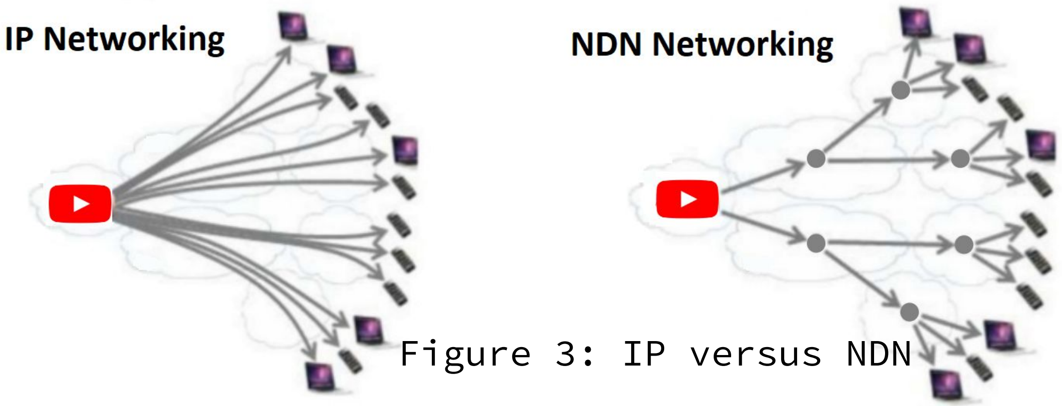


Figure 3: IP versus NDN

PID types

PID Types	PID Type Identifier	Delimiter	Authority	Delimiter	Name
URN	urn	:	<NID>	:	<NSS>
HANDLE	handle	:	<Handle Naming Authority>	/	<Handle Local Name>
DOI	doi	:	10.<Naming Authority>	/	<doi name syntax>
ARK	ark	:	/<NAAN>	/	<Name>[<Qualifier>]

Related work

- Rahaf Mousa
  - Focused on DOI > NDN
    - Concluded that PID > NDN is possible
  - Most optimal caching strategy in NDN
- Andreas Karakannas
  - For every PID type a PID > NDN mapping server
  - States:
    - "PID > NDN mapping will be highly depended on the clients NDN browser which will need to be updated every time new rule would be appeared or changed"
- Spiros Koulouzis et al.
  - NaaS4PID
    - Supports one PID type

PID → NDN namespace interoperability

- Translation is transparent to the user
- Support for multiple PID types
- Extensible with future PID types with different naming schemes

Handle:	[http://hdl.handle.net/]**20/5000/481/objects/example_object**
NDN:	**/ndn/handle/20/5000/481/objects/example_object**
URN:	[http://resolver.kb.nl/resolve?urn=]**anp:1938:10:01:2:mpeg21**
NDN:	**/ndn/urn/anp/1938/10/01/2/mpeg21**

PID → NDN model

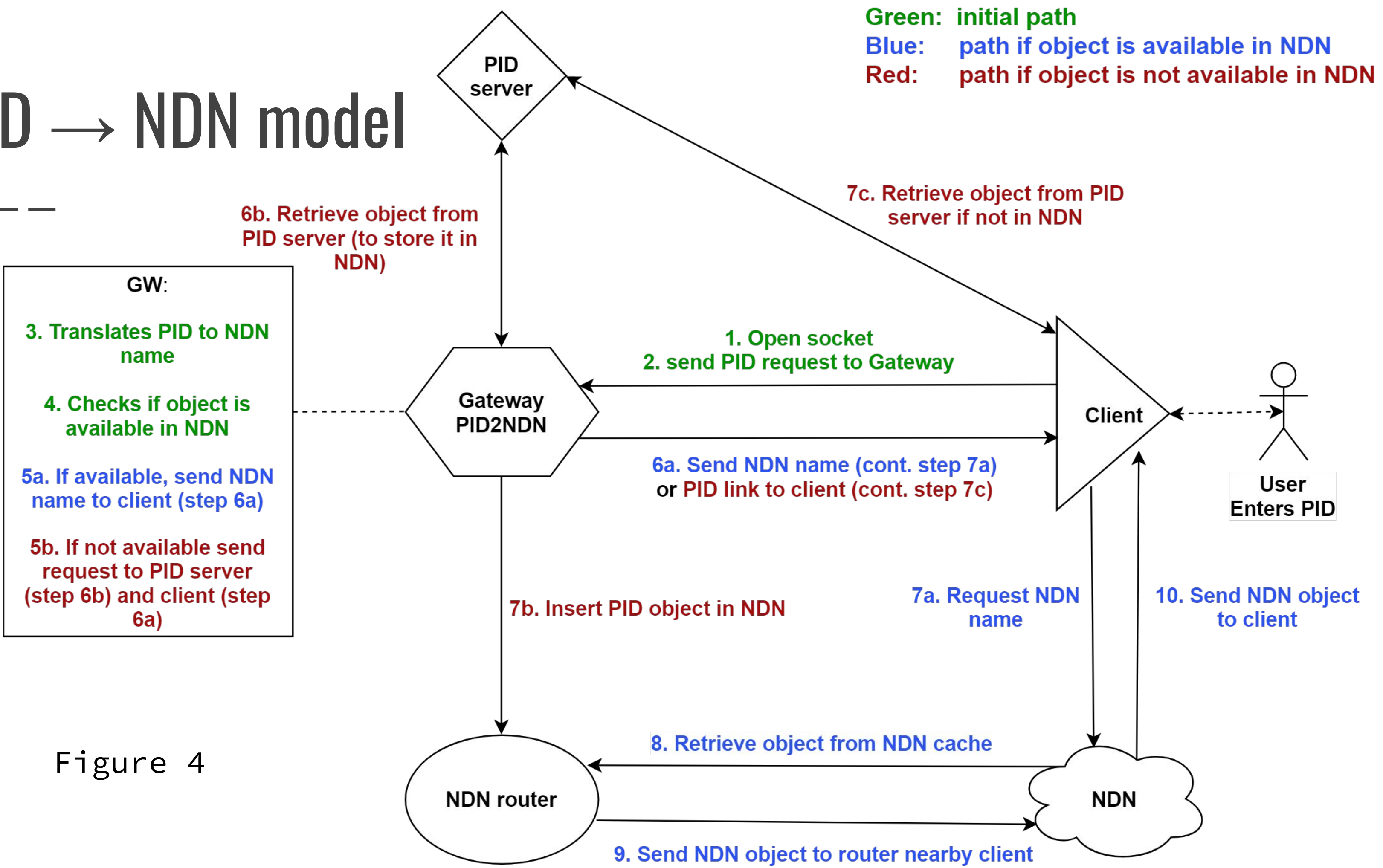


Figure 4

Proof of concept

How to make NDN scalable and software definable?

- Kubernetes
  - Open-source container-orchestration system
    - Deployment
    - Scaling
    - Management
- SDN-style control
  - Centrally deploy and configure containers (NDN functions)
    - Add roles (routers)
    - Configure routes
    - Allocate resources

Architecture drawing - Proof of concept

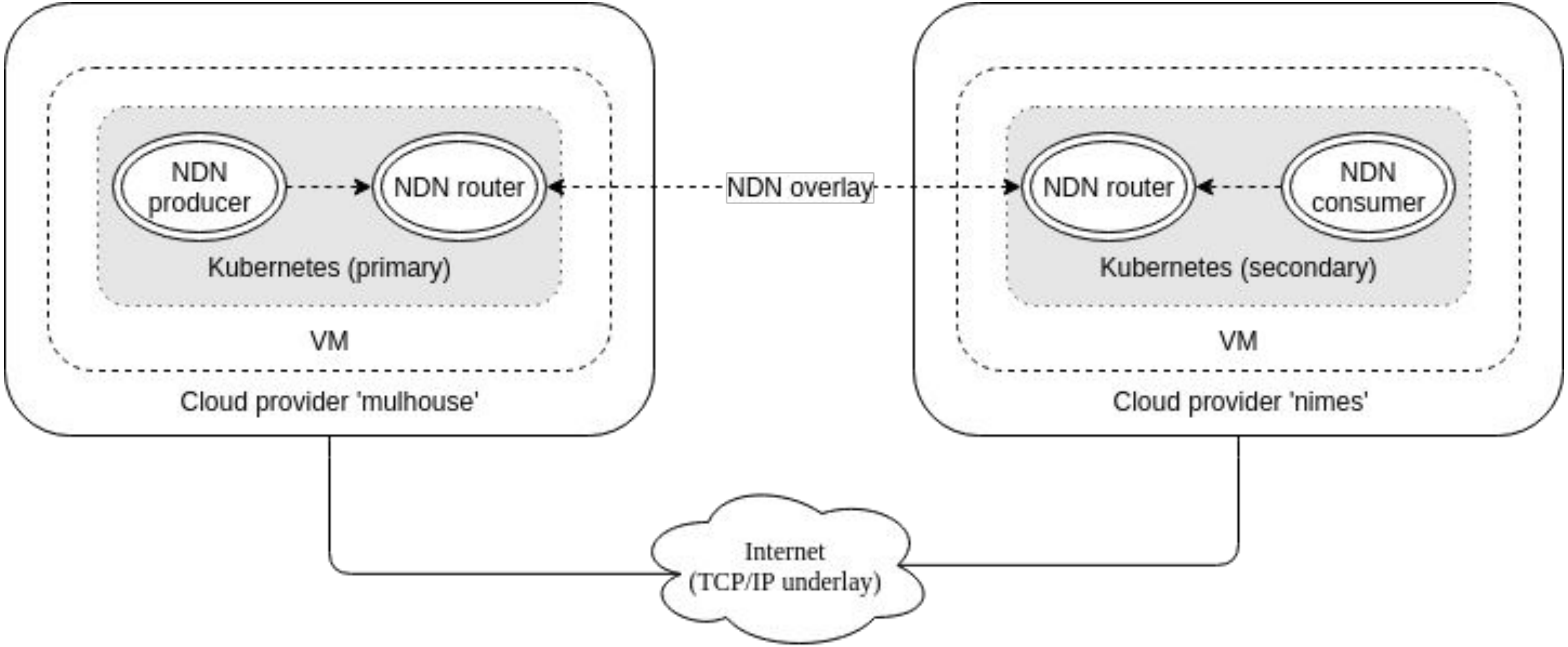


Figure 5

How to plan the NDN network

- The challenge becomes
  - How to manage/plan/deploy such a diverse infrastructure?
- Single description to plan and deploy needed
  - Is there an open standard available?

How to plan the NDN network (TOSCA)

- What is TOSCA?
  - Topology and Orchestration Specification for Cloud Applications
  - Declarative Domain Specific Language (YAML/XML)
  - TOSCA descriptions → orchestrator
  - Used to describe complete lifecycle
    - **Hosts** (bare metal, VM, containers)
    - **Software components** (applications, databases, middleware)
    - **Network components** (load balancers, gateways, VNF’s)
- TOSCA is agnostic towards orchestrators
  - DRIP
  - OpenStack
  - And gaining popularity

Different types in TOSCA to describe building blocks

- Eight different types to use
    - **Node**
    - **Relationships**
    - Artifacts
    - Capabilities
    - ***Interface***
    - Groups
    - Policies
    - Data

- Node
    - Host, **container**, **VM**, etc.
- Relationships
    - Connects nodes to each other
    - **dependsOn**, hostedOn, connectsTo
- Interface
    - Set of hooks
    - Actions to: **Create**, configure, start, stop or delete

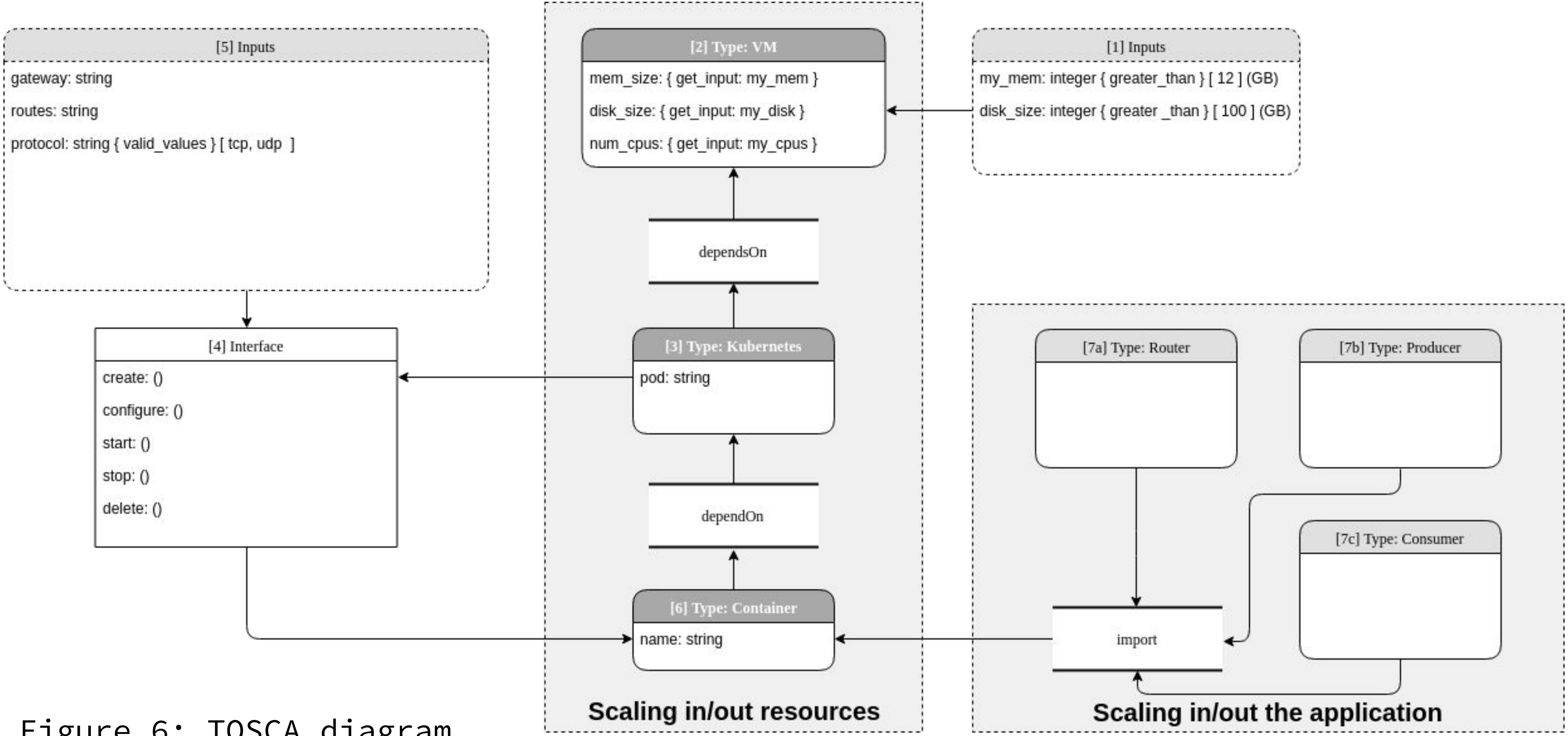


Figure 6: TOSCA diagram

How to make NDN software definable? (Kubernetes)

```yaml
spec:
  hostname: ndn-router-1
  nodeName: mulhouse
  containers:
    - image: aqua1te/ndn:router3
      name: ndn-router1
      env:
        - name: gateway
          value: ndn-producer-2
        - name: routes
          value: /ndn/handle /ndn/ark
        - name: protocol
          value: tcp
```

Activities   Terminal ▾   wo jul 3 17:44

RP2

```
[kjong@defiant Kubernetes (master)]$ kubectl apply -f ex
panded-cluster.yml
```

Cluster control

```
Every 2,0s: kubectl get pods -o wide                                    defiant: Wed Jul  3 17:44:31 2019
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
ndn-consumer-1	1/1	Running	0	8m34s	10.244.0.251	mulhouse	<none>	<none>
ndn-consumer-2	1/1	Running	0	8m34s	10.244.1.90	nimes	<none>	<none>
ndn-producer-1	1/1	Running	0	8m34s	10.244.0.252	mulhouse	<none>	<none>
ndn-router-1	1/1	Running	0	8m34s	10.244.0.253	mulhouse	<none>	<none>
ndn-router-2	1/1	Running	0	8m34s	10.244.1.91	nimes	<none>	<none>

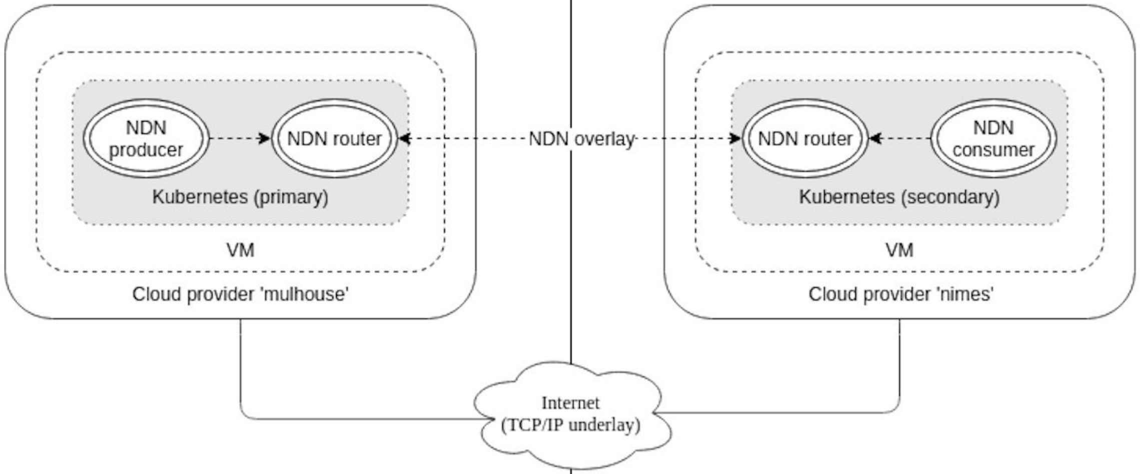
Cluster status

```
[kjong@defiant ~]$ kubectl exec -it ndn-consumer-1 bash
```

Consumer 1
NDN -> PID -> NDN

```
[kjong@defiant ~]$ kubectl exec -it ndn-consumer-2 bash
```

Consumer 2
NDN

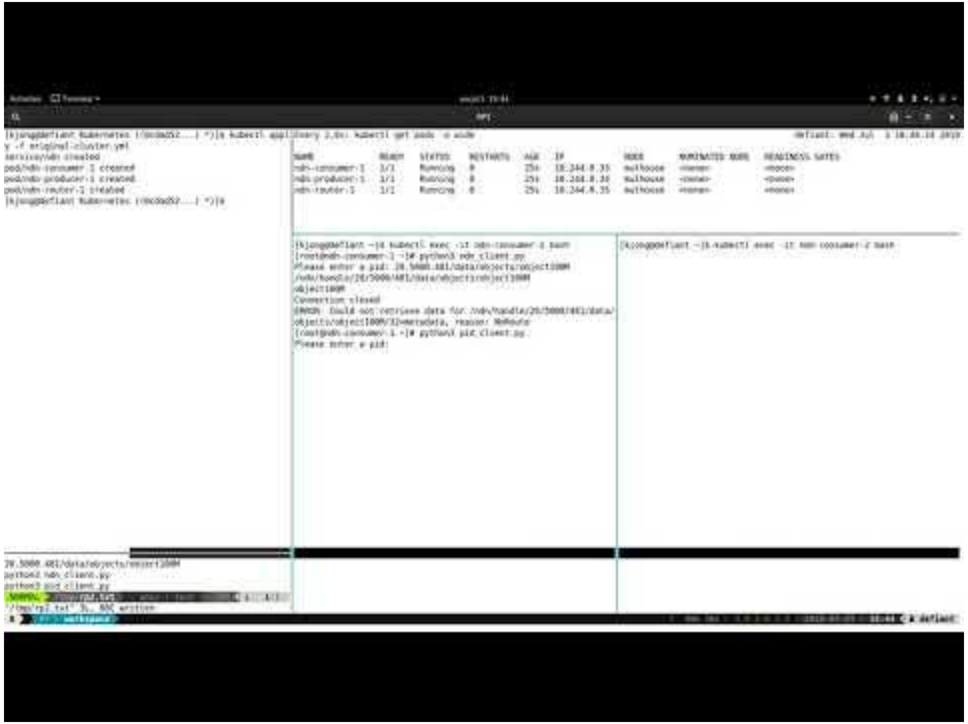



```
# Handle URI
20.500.481/data/objects/object100M

# Client scripts
python3 ndn_client.py
python3 pid_client.py
~
~
~
~
NORMAL  /tmp/pid.txt  unix  text  100%  6:21
"/tmp/pid.txt" [New] 6L, 110C written
0  0*  rp2     3h 3m 19s  1.2 1.0 1.0  2019-07-03  17:44  defiant
```

Copy area

Demo



Conclusion

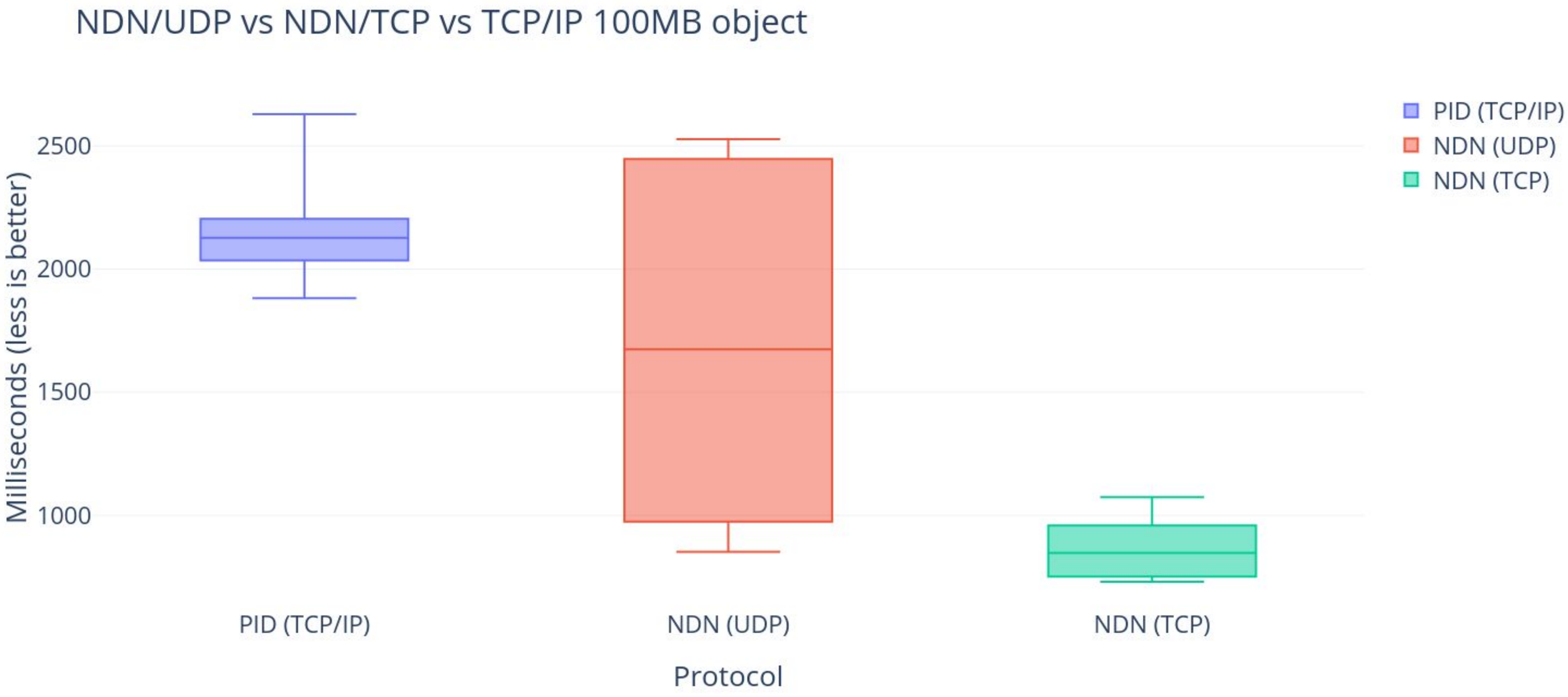
- Deployment planning
  - TOSCA can describe complete lifecycle of infrastructure
- Easy scaling out to other clouds
  - VM’s used to allocate/deallocate resources in the cloud
  - Kubernetes used to scale in/out the application (NDN)
  - Bringing data closer to the user decreases latency and chance of congestion
- Interoperability between different PID types is possible
  - Adding new PID types is low effort cost

Future work

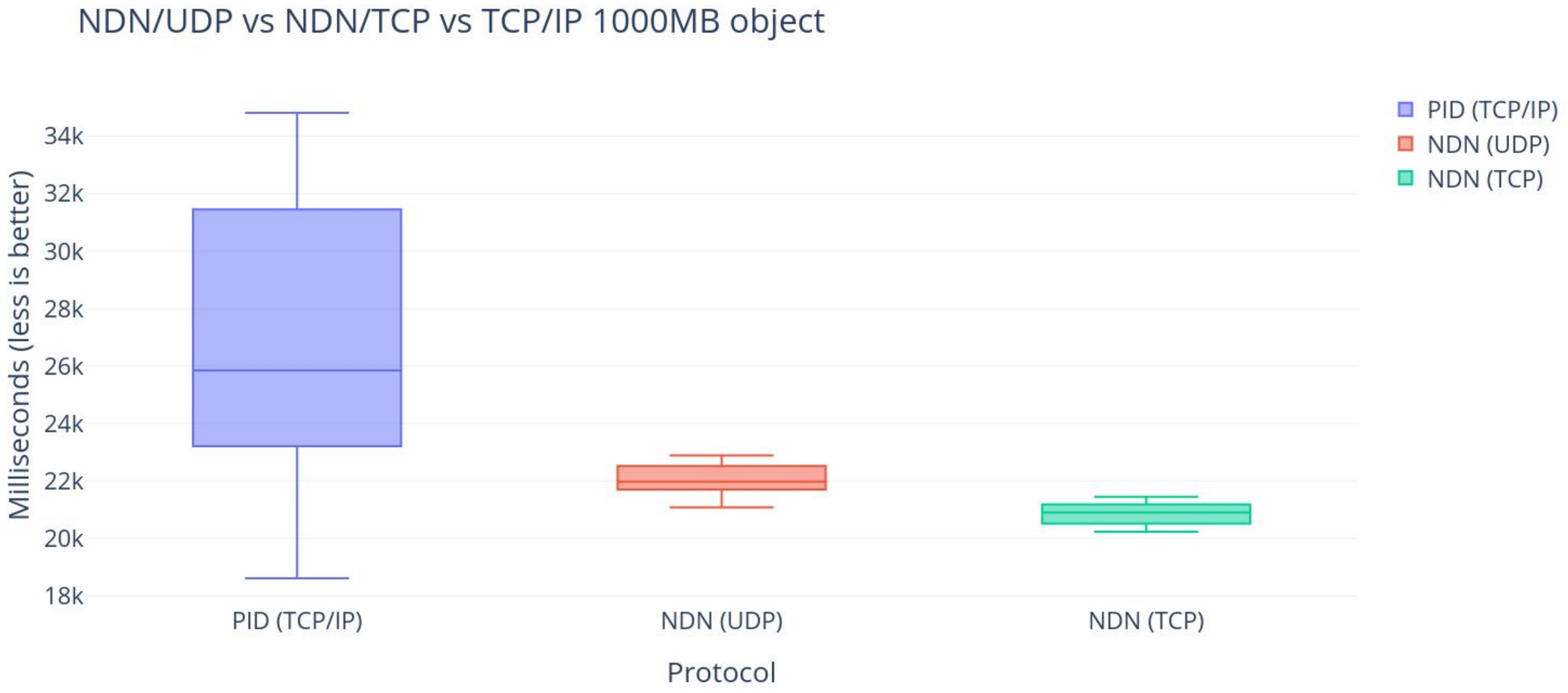
- TOSCA blueprints are conceptual
  - The VM and Kubernetes was deployed manually
  - Full implementation developed needed with an orchestrator such as e.g. DRIP
- NDN is still experimental
  - Explore performance bottlenecks (benchmarking)
  - Test routing protocols (e.g. OSPFN)
- Extent Kubernetes with intelligence
  - Where to deploy NDN routers (containers)?
- Incorporate the PID → NDN translation into NDN software natively

Questions?

Performance of proof of concept setup

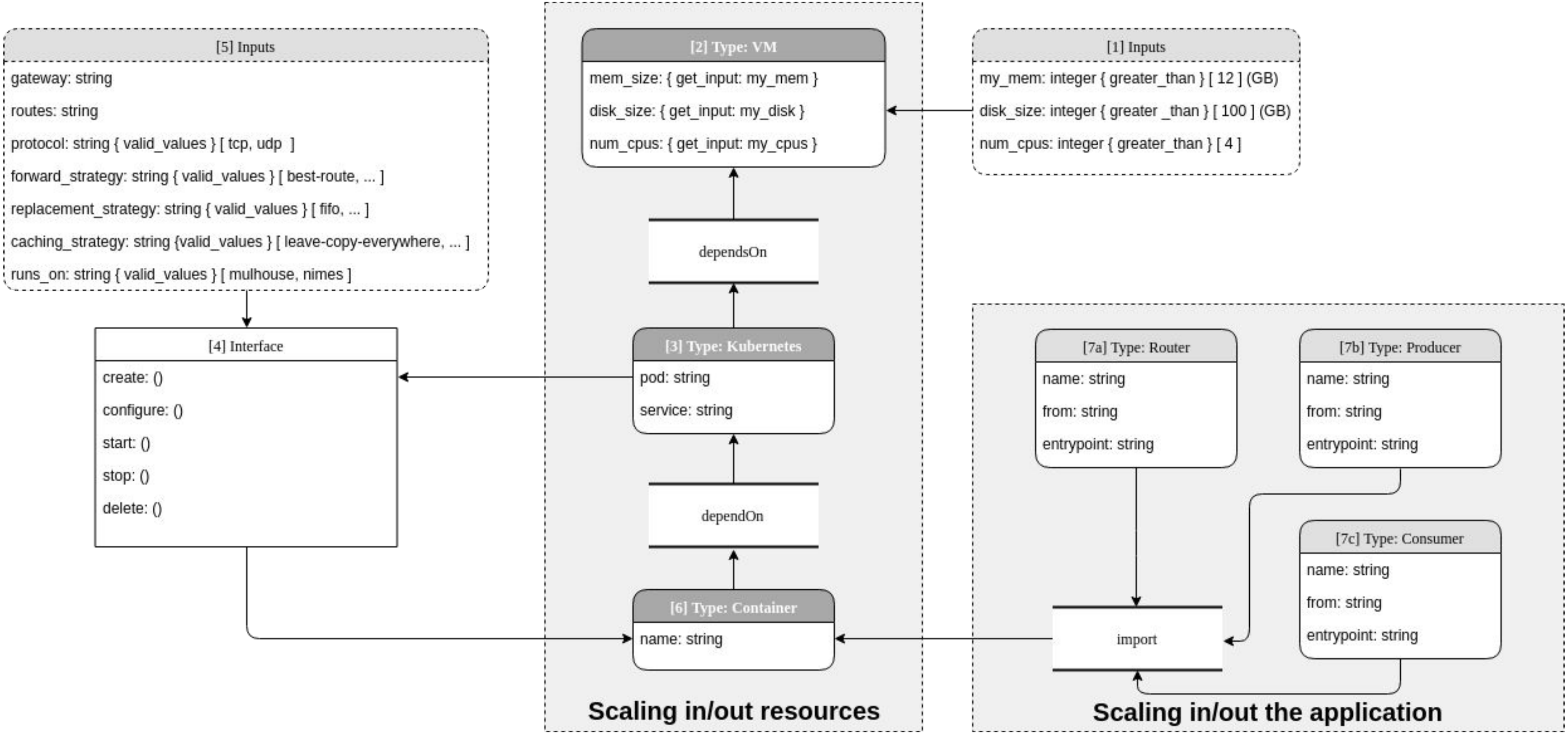


Performance of proof of concept setup



Performance of proof of concept setup

- Difference in percentage
  - 100MB file:
    - NDN (UDP) vs PID (TCP/IP): 27%
    - NDN (TCP) vs PID (TCP/IP): 150%
    - NDN (TCP) vs NDN (UDP): 98%
  - 1000MB file:
    - NDN (UDP) vs PID (TCP/IP): 18%
    - NDN (TCP) vs PID (TCP/IP): 24%
    - NDN (TCP) vs NDN (UDP): 5%


[5] Inputs
gateway: string
routes: string
protocol: string { valid_values } [tcp, udp]
forward_strategy: string { valid_values } [best-route, ...]
replacement_strategy: string { valid_values } [fifo, ...]
caching_strategy: string {valid_values } [leave-copy-everywhere, ...]
runs_on: string { valid_values } [mulhouse, nimes]
[4] Interface
create: ()
configure: ()
start: ()
stop: ()
delete: ()
[2] Type: VM
mem_size: { get_input: my_mem }
disk_size: { get_input: my_disk }
num_cpus: { get_input: my_cpus }
dependsOn
[3] Type: Kubernetes
pod: string
service: string
dependOn
[6] Type: Container
name: string
Scaling in/out resources
[1] Inputs
my_mem: integer { greater_than } [12] (GB)
disk_size: integer { greater _than } [100] (GB)
num_cpus: integer { greater_than } [4]
[7a] Type: Router
name: string
from: string
entrypoint: string
[7b] Type: Producer
name: string
from: string
entrypoint: string
[7c] Type: Consumer
name: string
from: string
entrypoint: string
import
Scaling in/out the application

NDN performance bottlenecks

- Underlay (TCP/IP)
  - UDP vs TCP
  - MTU sizes
- Processing problems in software
  - Slow packet decode functions (35.4% time spend on decoding)
  - Long names can degrade performance
- Named data forwarding scaling
  - Routing table sizes
  - Forward strategies
- Named data caching scaling
  - Cache strategies + size
    - LCE (Leave Copy Everywhere)
  - Cache replacement strategies