Docker, Clustering, Scalability
A Case Study

John Quick

Smartvox Limited

The Design Specification

SIP Registrar/Proxy in front of a FreeSwitch-based core

Docker-based (Bare metal hosting)

Scalable (Microservices concept)

Short Term Plans:

- WebRTC support
- Deploy using Kubernetes

Longer Term Plans: Multi site deployment

Strategy

Identify gaps in my knowledge

Search the web for ideas - reports, articles, blogs

Read product documentation

Set up a test lab

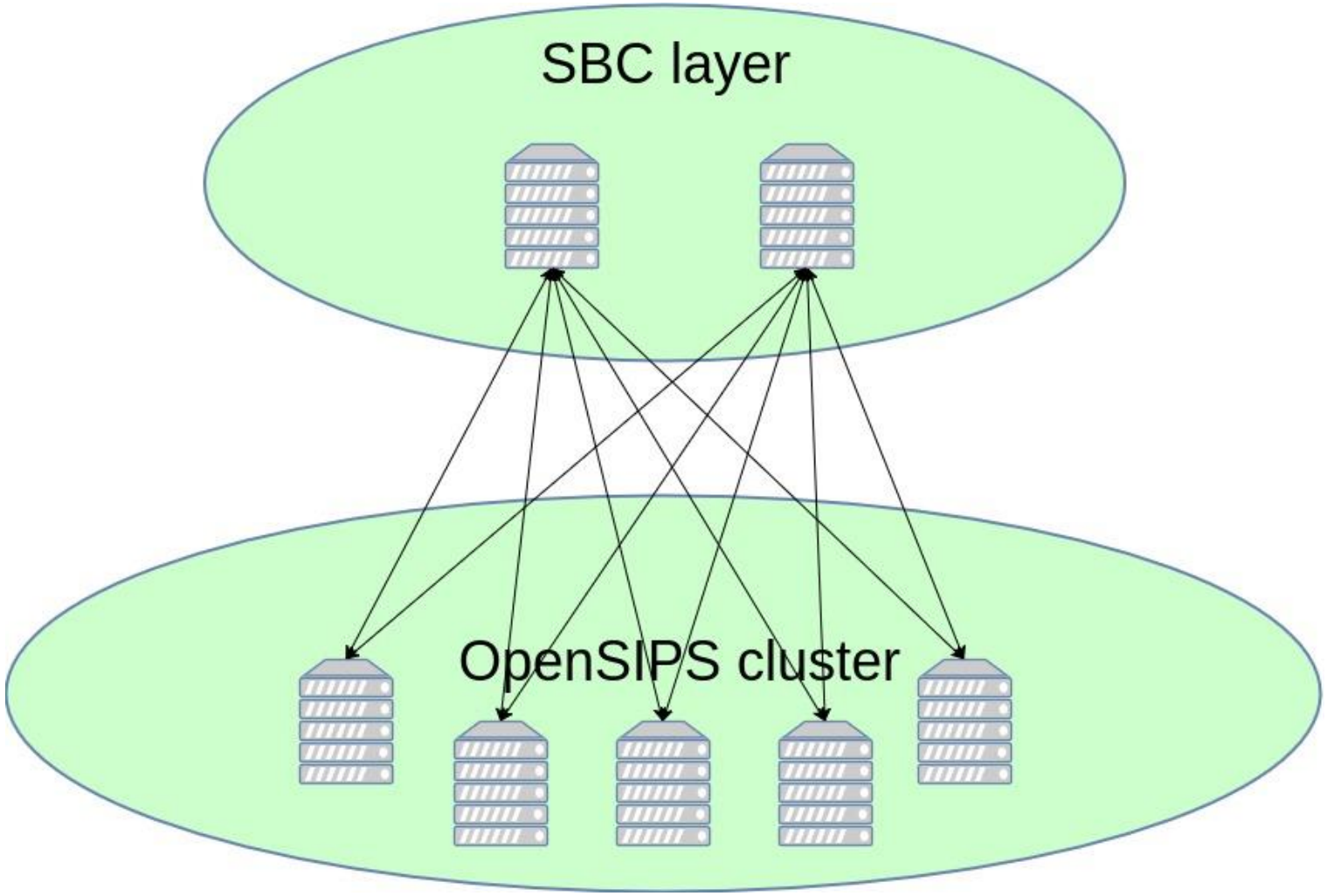
Learn about concepts through testing

Identify issues and snags through testing

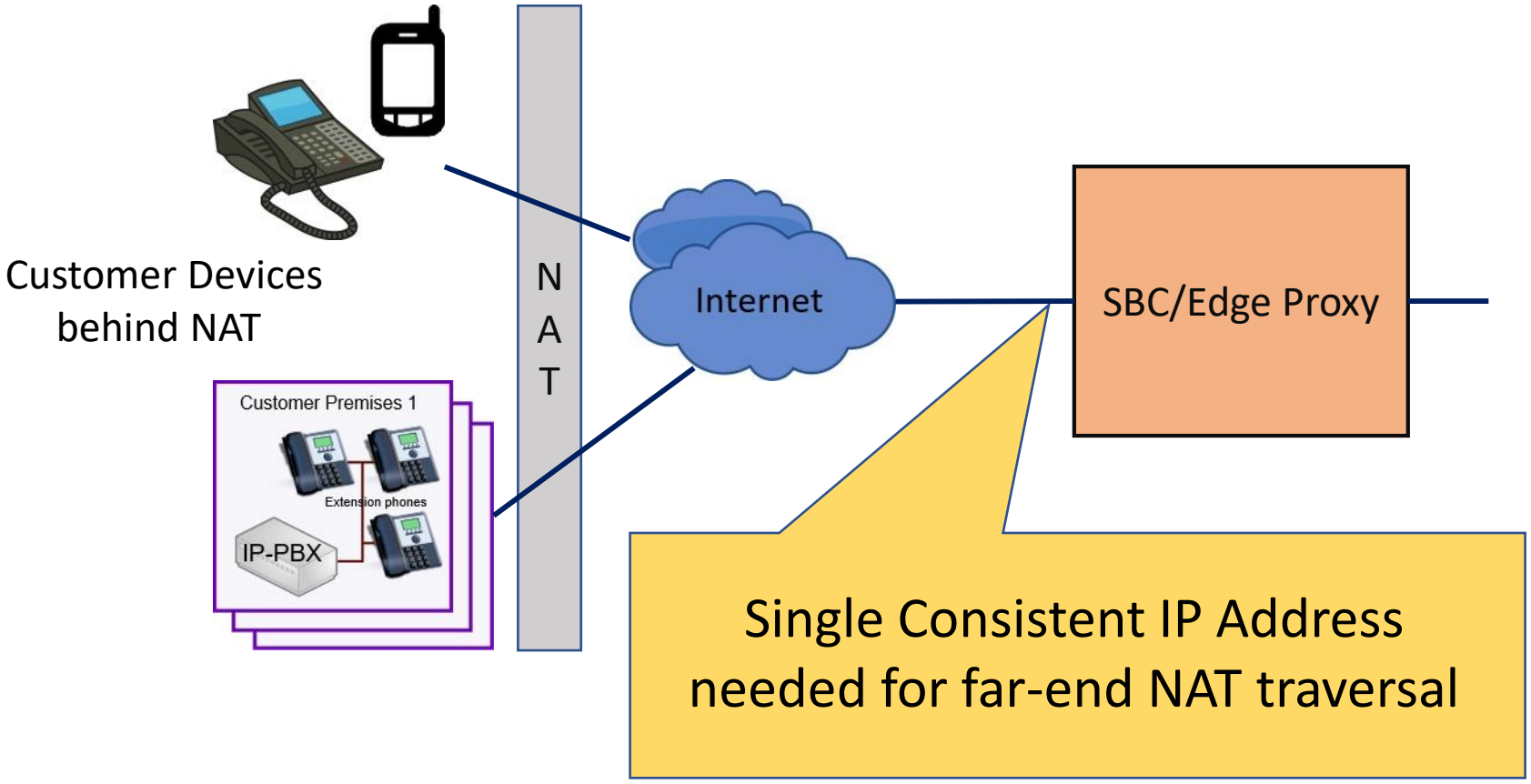
Key Topics

- Docker
- Clustering topologies
- Load sharing
- WebRTC
- Kubernetes

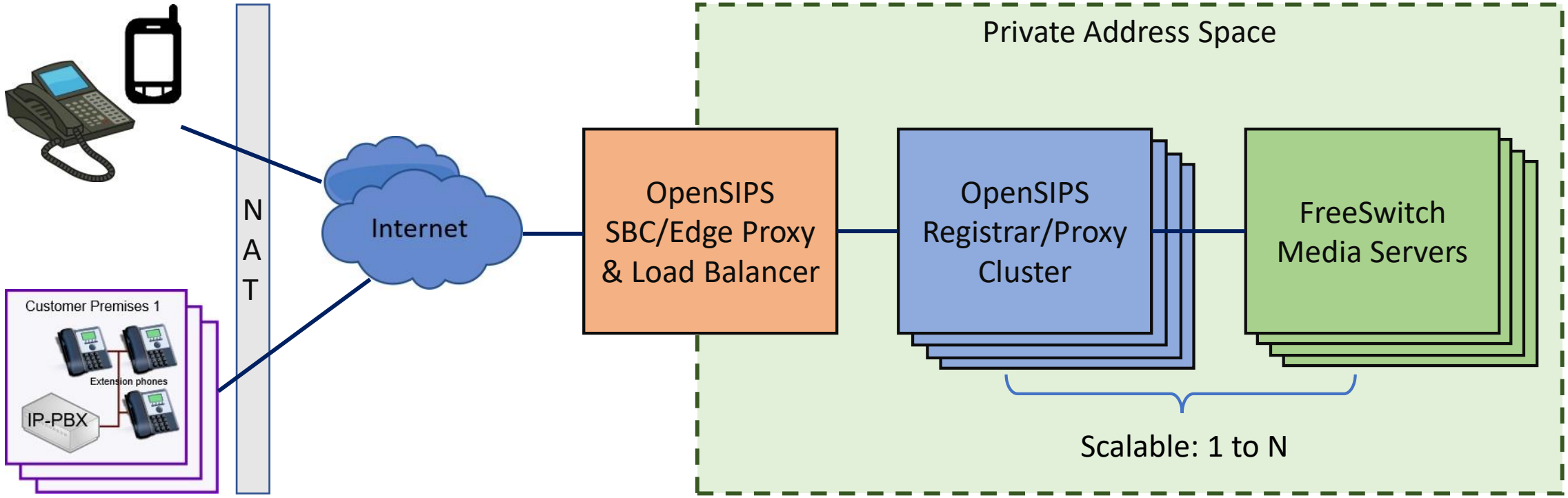
Clustering Topology: Initial Design Inspiration



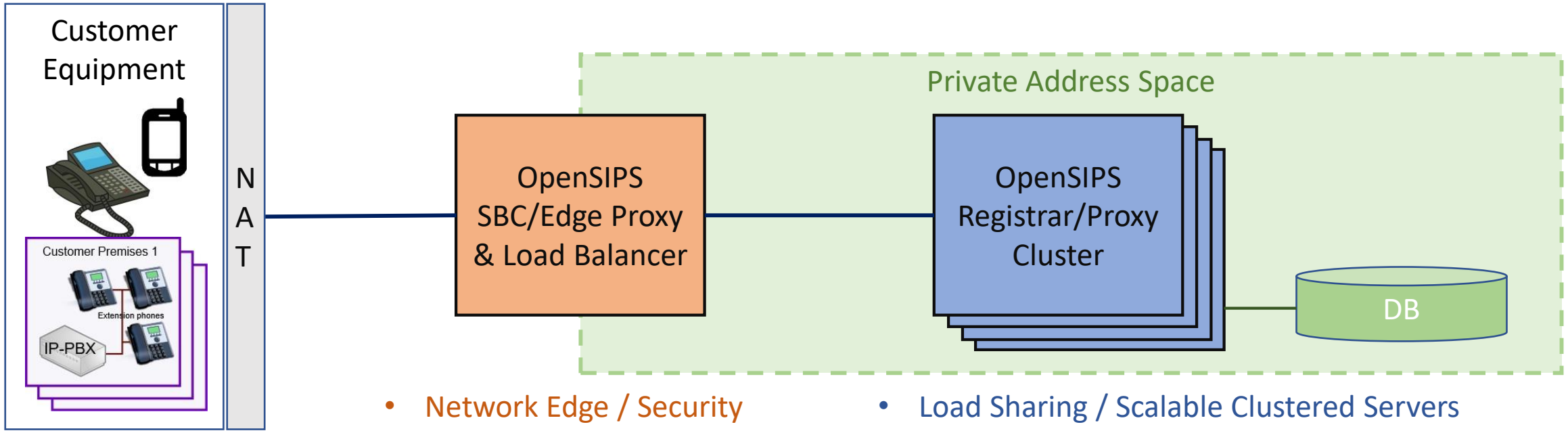
Why do we need the SBC / Edge Proxy?



Clustered servers behind an Edge Proxy



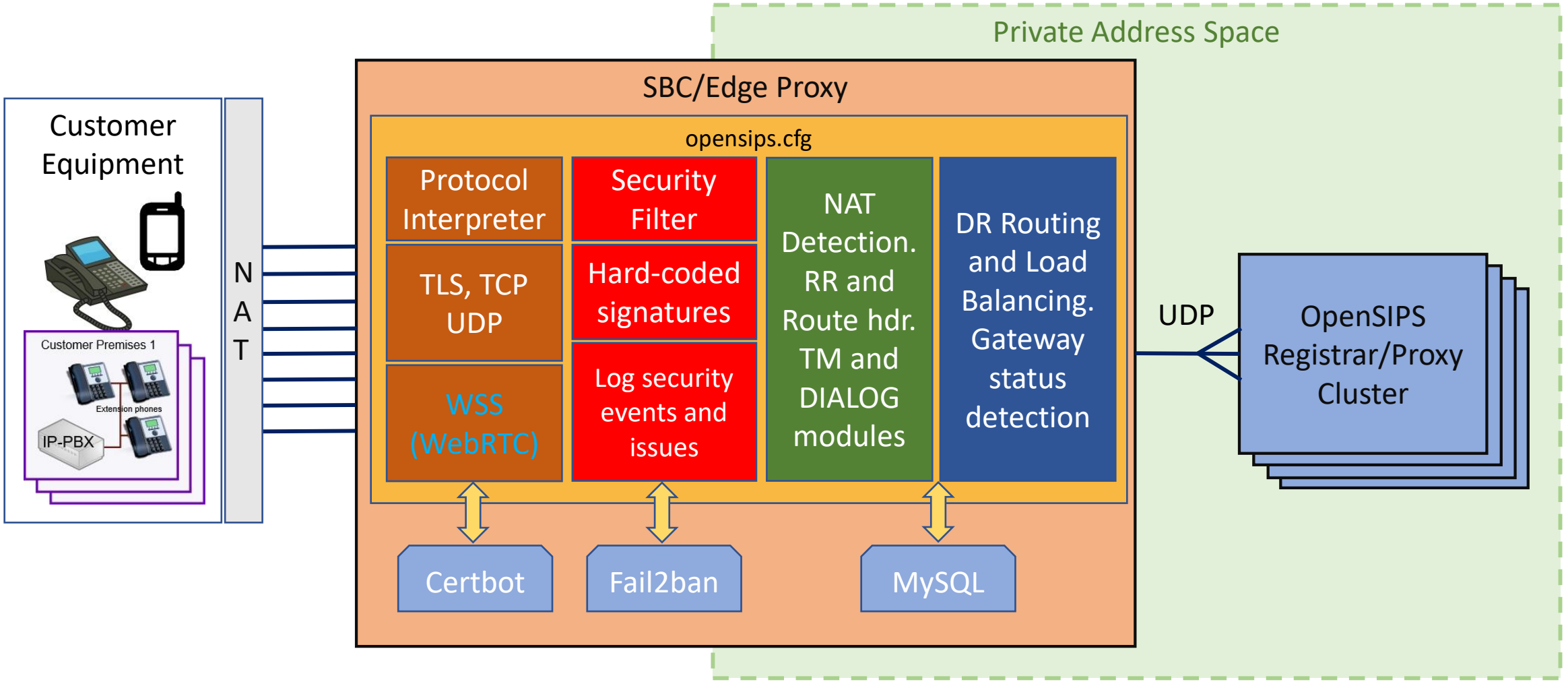
OpenSIPS Server Roles



- Network Edge / Security
- Fast and Lightweight
- Resilient (Highly Available)

- Load Sharing / Scalable Clustered Servers
- Responsible for “heavy lifting” including
  1. User Device Registrations
  2. Authentication

SBC / Edge Proxy - deconstructed



Risks when using SBC / Edge Proxy

- Single Point of Failure

  HA Cluster using Pacemaker

  BGP Network Connectivity

  Future options – Dual DC, Anycast

- Traffic Bottleneck

  Enable DNS Caching (module DNS_CACHE)

  Optimise settings in opensips.cfg:
  - open_files_limit
  - children (worker processes)

  Optimise and monitor pkmem & shmem

  As Stateless as possible

As Stateless As Possible

- Problem:

Correct routing of sequential requests with minimal memory use

- Solution

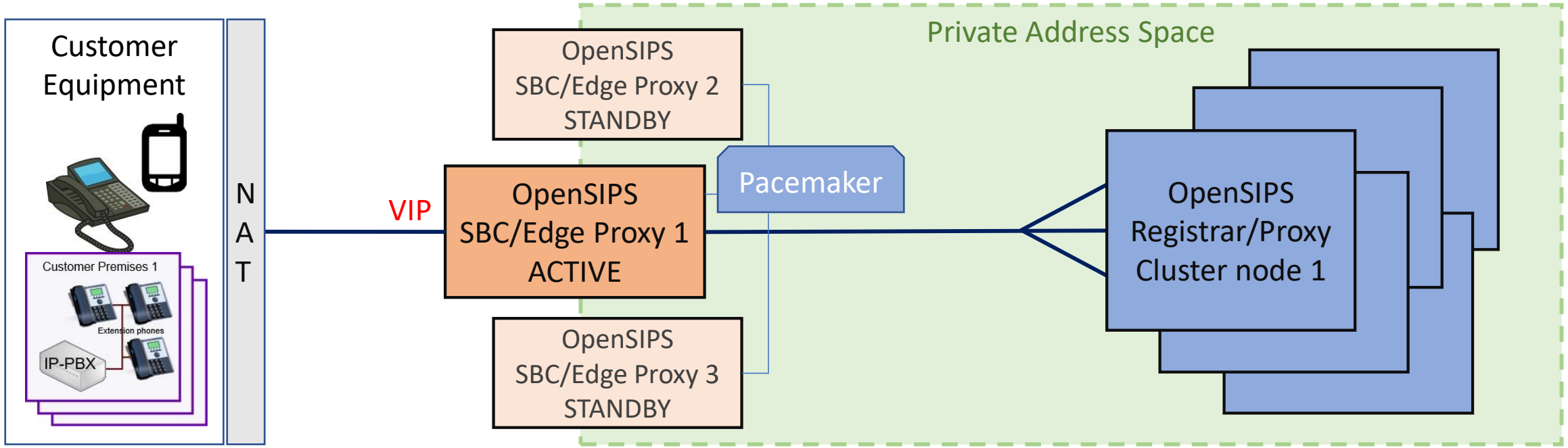
a) Record-Route, Route and Contact headers
b) Don’t use Topology Hiding module

- Still to be resolved

TM and DIALOGUE modules

a) Memory overhead?
b) Are they even needed?

OpenSIPS: Application of Clusterer Module



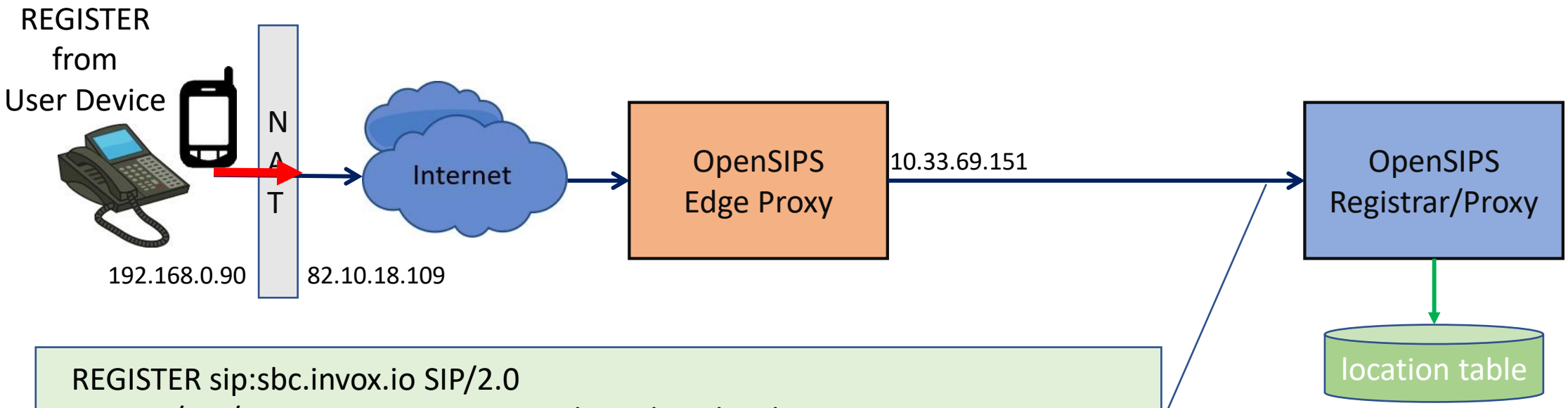
- Shared DIALOG data
- Pacemaker controls VIP

- USRLOC Full Sharing Cluster
- NAT Ping Sharing
- Scalable 1 to N Cluster Nodes

Topology-related Issues

Using Path headers for NAT'd user devices - 1

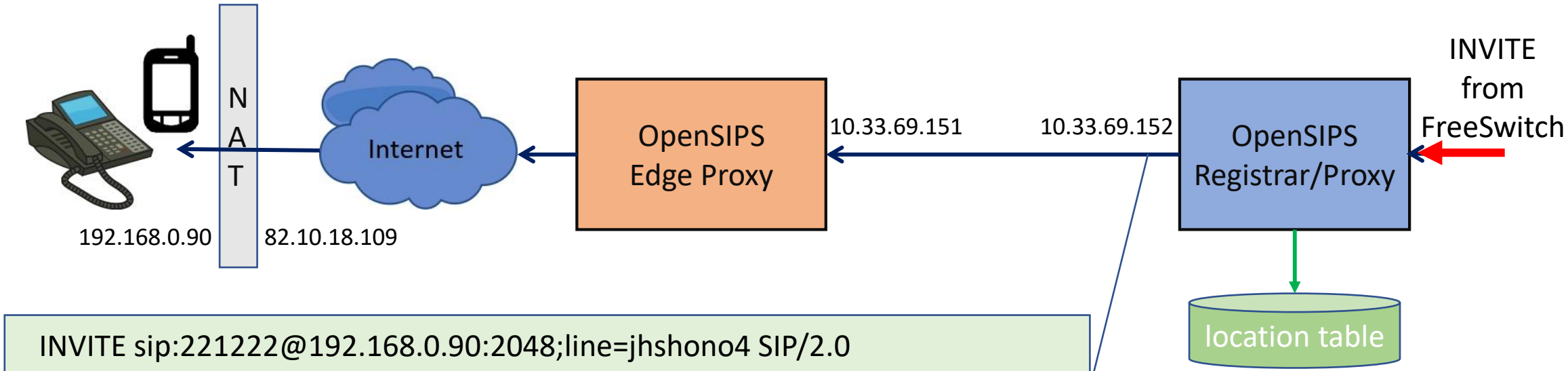
- Path Header is added to REGISTER request by our Edge Proxy
- Includes "received=" parameter when user device behind NAT



REGISTER sip:sbc.invox.io SIP/2.0
Via: SIP/2.0/UDP 10.33.69.151:5060;branch=z9hG4bK1351
Via: SIP/2.0/UDP 192.168.0.90:2048;received=82.10.18.109;branch=z9hG4bK.
Contact: <sip:221222@192.168.0.90:2048;line=jhshono4>
Path: <sip:mysbc@10.33.69.151;lr;received=sip:82.10.18.109:12202>

Using Path headers for NAT'd user devices - 2

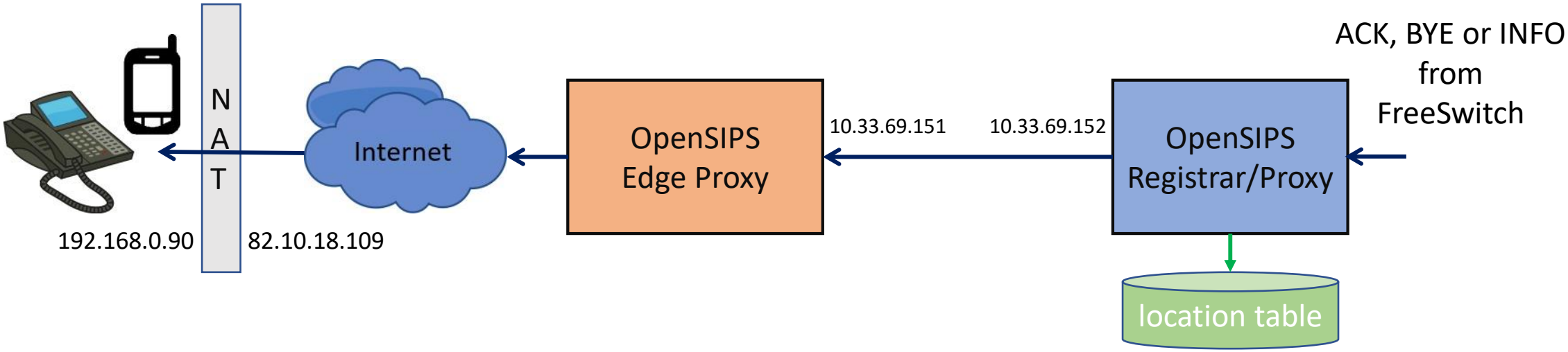
- In the Registrar Proxy, lookup() retrieves data from location table
- Registrar Proxy automatically adds a Route header based on Path



INVITE sip:221222@192.168.0.90:2048;line=jhshono4 SIP/2.0
Record-Route: <sip:10.33.69.152;lr;ftag=4eUm..;natua=10.33.69.152>
Via..
Route: <sip:mysbc@10.33.69.151;lr;received=sip:82.10.18.109:12202>
Other headers..

Using Path headers for NAT'd user devices - 3

- Sequential Requests such as ACK and BYE also need to call lookup



- If no lookup() in the Registrar/Proxy, there is no "received=" parameter
- No "received=" parameter means Edge Proxy cannot route Request

Using Path headers for NAT'd user devices

- Problem:

Edge Proxy must always receive a Route header with "received=" parameter when the request is going to a NAT'd customer device

- Solution

a) During device registration, detect NAT and add a branch flag which gets stored in the relevant location table record

b) During initial INVITE, do lookup(). If the NAT branch flag is present then add parameter "natua" to the Record-Route hdr.

c) When handling sequential requests later, look for the "natua" parameter in the Route header. If found, do lookup() again

NAT’d user devices – code snippet

```
# Handle Loose_Routed packets
if (has_totag()) {
   if (loose_route()) {
      if (is_direction("downstream")) {
         if (check_route_param("natua=")) {
            remove_hf(“Route”);
            # Get best Path/Route info with location table lookup
            if (lookup("location"))
               xlog("L_WARN", "     Lookup worked R-URI=$ru  Src=$si\n");
            else
               xlog("L_WARN", "    !!Loose-Routed $rm request failed lookup!! \n");
         }
      }
```

Other Topology-related issues and snags

- Problem:

Load Balancer must send authenticated Requests to the same Registrar Proxy node that sent the Challenge

- Solution

a) Use local CacheDB as short term, fast access memory store
b) Load Balancing mechanism chooses node for initial request
c) Push address of selected node (use source address as key)
d) Request arrives with Digest Authentication
e) Pop node address back from local CacheDB

Other Topology-related issues and snags

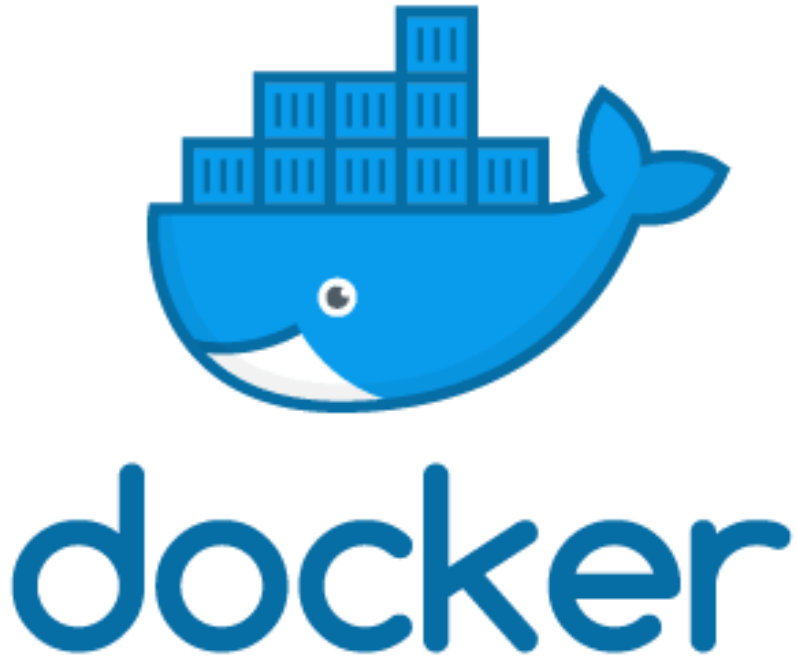
- Problem:

Customer devices dropping oversized INVITE requests;

MTU exceeded due to addition of Record-Route and Via headers

- Solution

a) Strip out unnecessary extra headers added by FreeSwitch
b) Could use Compact Headers
c) Could use Topology Hiding, but only with a memory hit



Introducing Docker

A Crash Course

The Benefits of using Docker

- A great Deployment Tool
  - Portability
  - Repeatability
  - Lightweight and Fast
- Encourages Good Practice
  - Microservices architecture
  - Well defined environment – O/S, Packages, Data, Configuration
  - Well defined build sequence
- A Basis for Scalability
  - PaaS – e.g. Kubernetes
  - IaaS – e.g. Amazon AWS

Some Docker basics

Image

- A file defining what will be in the container when it runs
- Official “base” images are available from Docker Hub

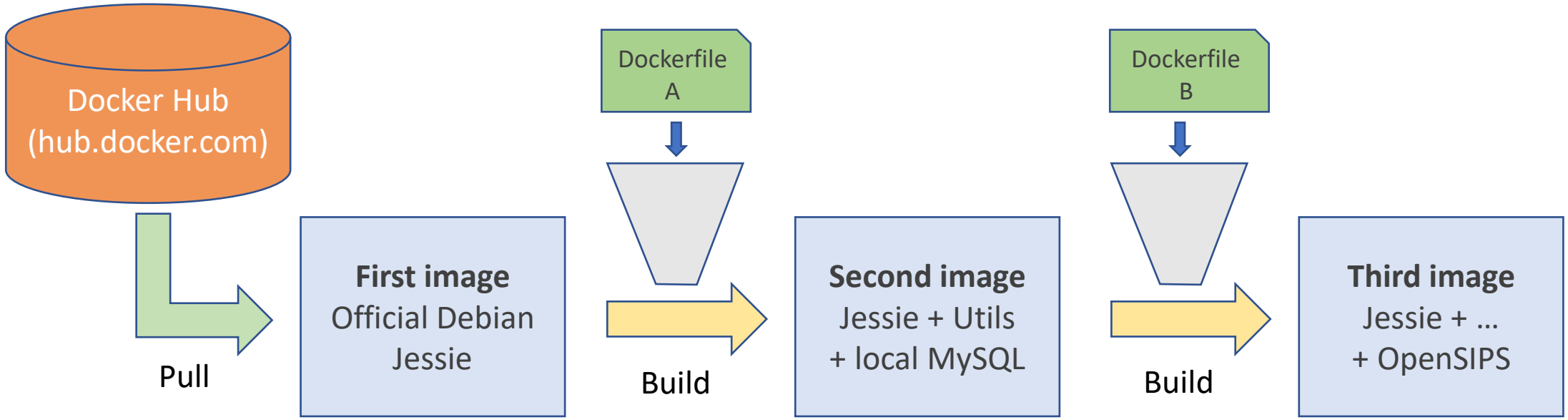
Docker Hub (hub.docker.com)

- A registry of publicly available Images – similar to github
- Pull images from Docker Hub into your own local store
- Push images to Docker Hub. Private storage areas are possible

Some Docker basics

Building your own Images

- New images are built, layer-by-layer, on top of existing one



Example of a Docker build file

```
FROM invoxio/opensips-base-deb:v2.4.4
LABEL maintainer="john.quick@smartvox.co.uk"
WORKDIR /etc/opensips

# Copy the working cfg files for our OpenSIPS Server
COPY *.cfg /etc/opensips/

# Copy the service start bash script file to the image (overwrites any pre-existing version)
COPY service-start.sh /usr/local/sbin/
RUN chmod +x /usr/local/sbin/service-start.sh

EXPOSE 5060 5678

CMD ["/bin/bash", "-c", "/usr/local/sbin/service-start.sh"]
```

Some Docker basics

Image

- A file defining what will be in the container when it runs
- Like a virtual machine ISO image, but much lighter weight
- Defines run-time environment, application and data

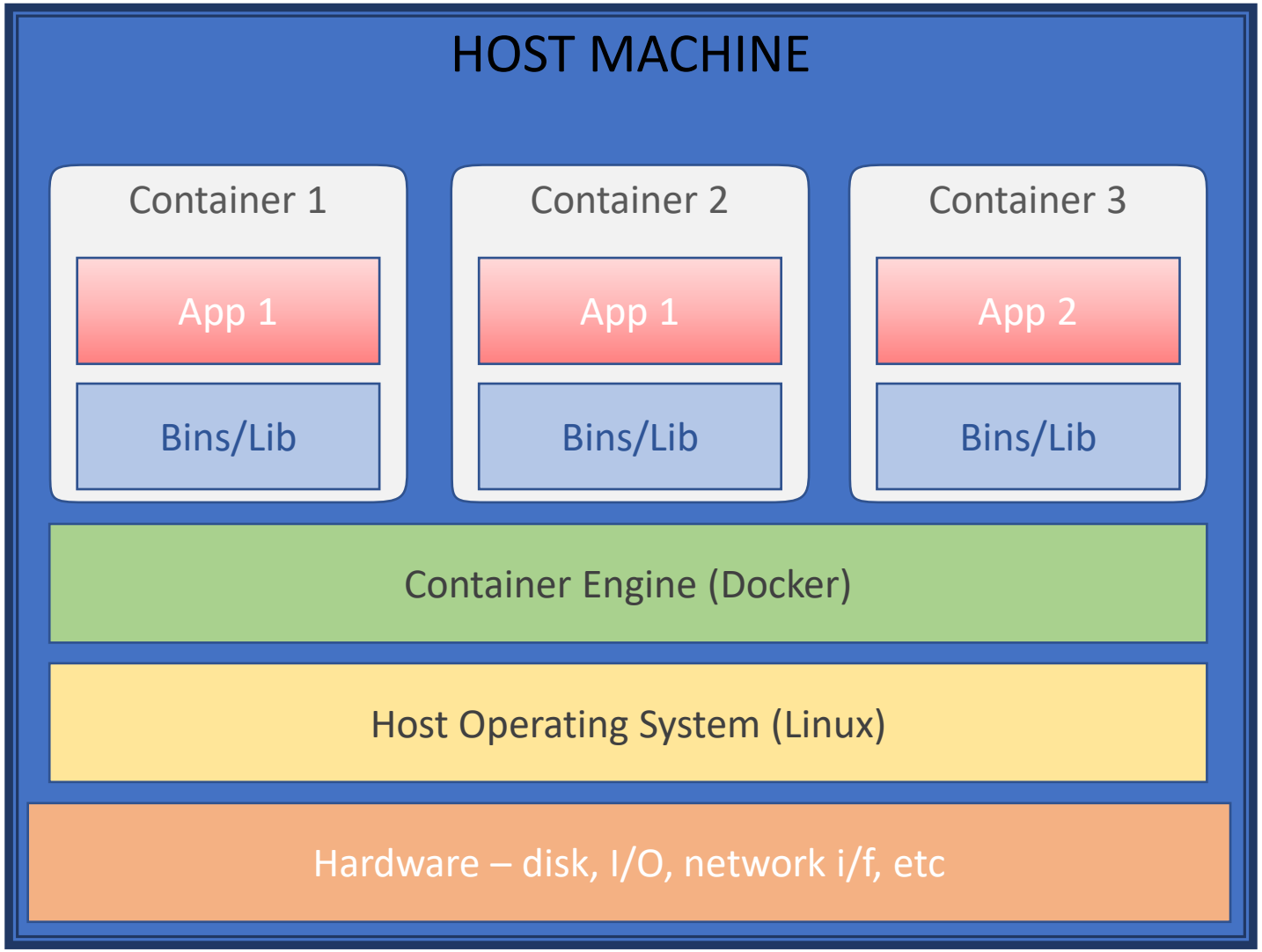
Containers

- Are instantiated Docker Images
- Encapsulate the run-time environment and a deployed application

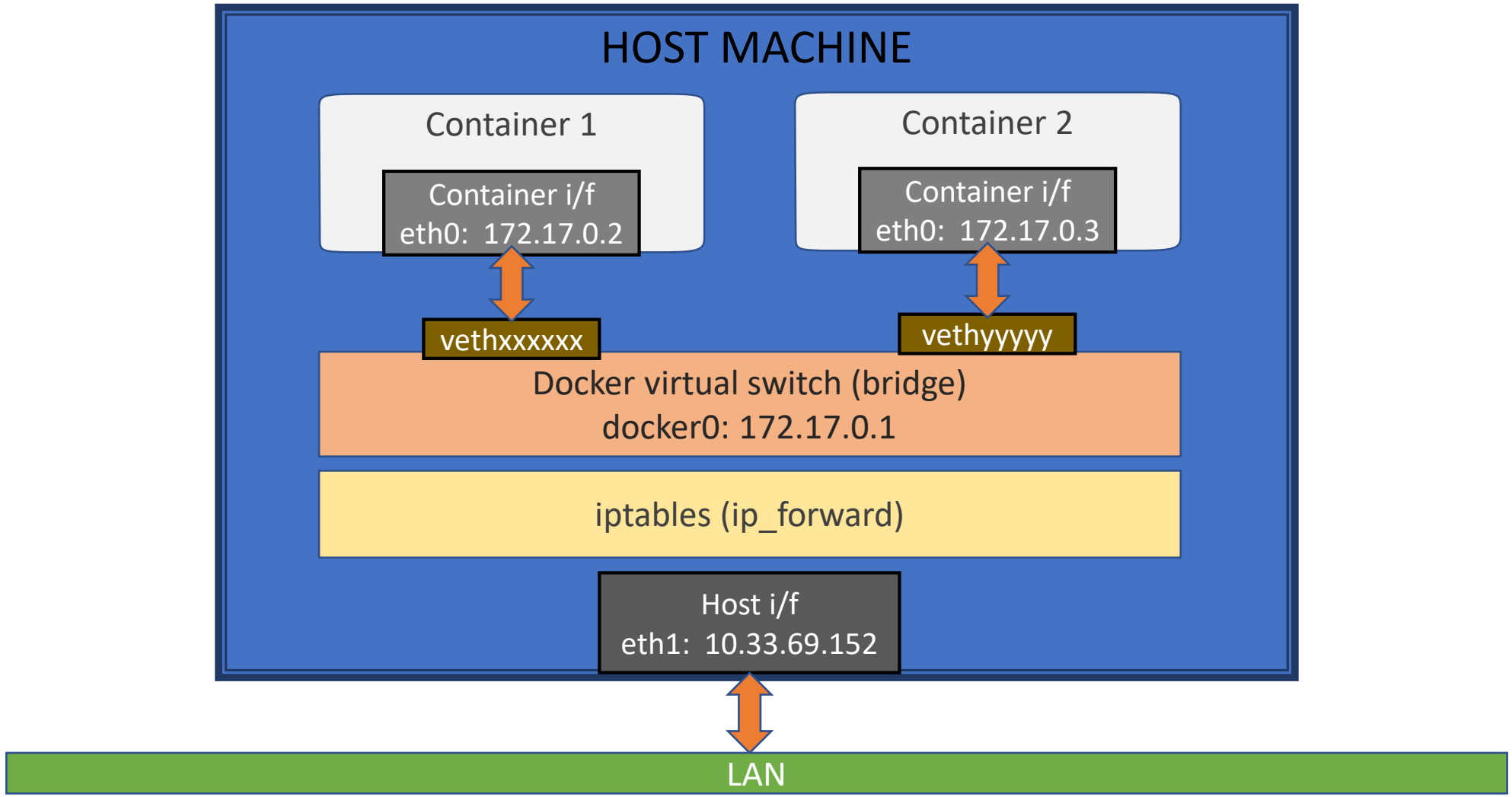
Docker basics – Running an image

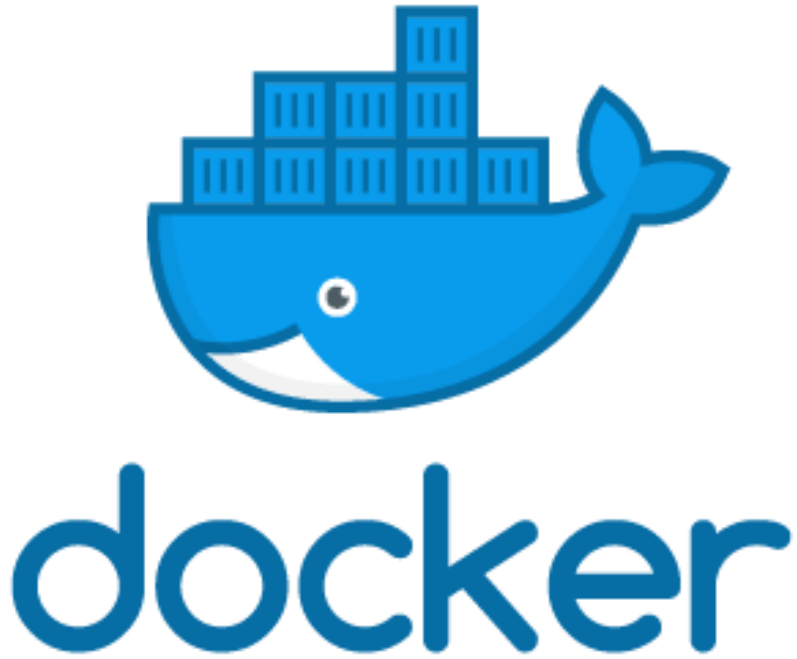
docker run	*..the command to run an image*
-dti	*..run in background with tty terminal*
--env-file=runenvparms	*..get parameter values from a file*
--name registrar	*..the container instance name*
-p 5060:5060/udp	*..expose ports*
invoxio/opensips-reg	*..use this image*

Docker architecture



Docker networking (default bridging mode)

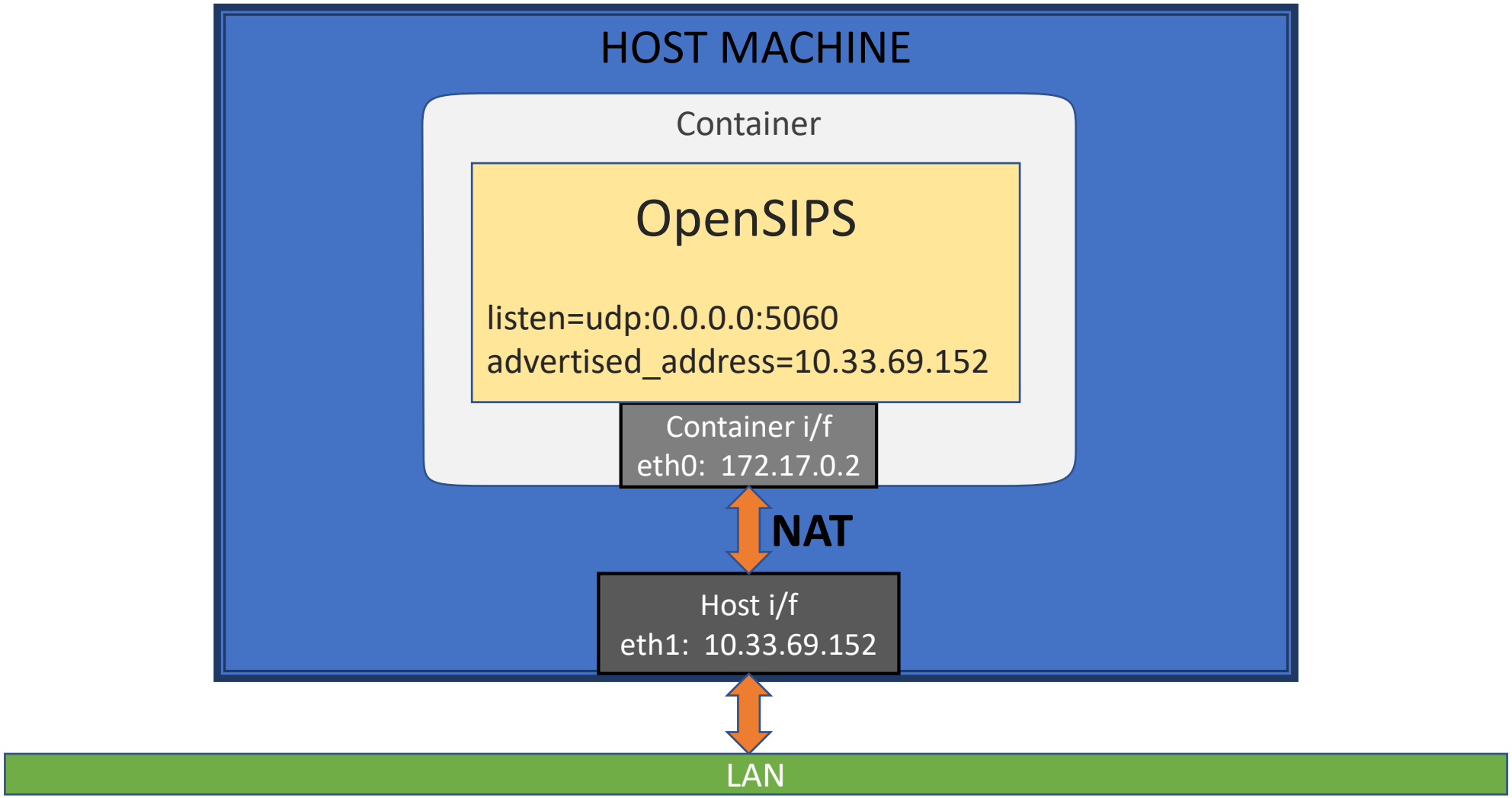




Using Docker with OpenSIPS

Practical Application

Docker networking: OpenSIPS behind NAT



Docker Containers as Microservices

Idealised scenario

- One Application per Container
- Application runs in the foreground
- Application start is the last action during start-up
- The container will stop if the application stops

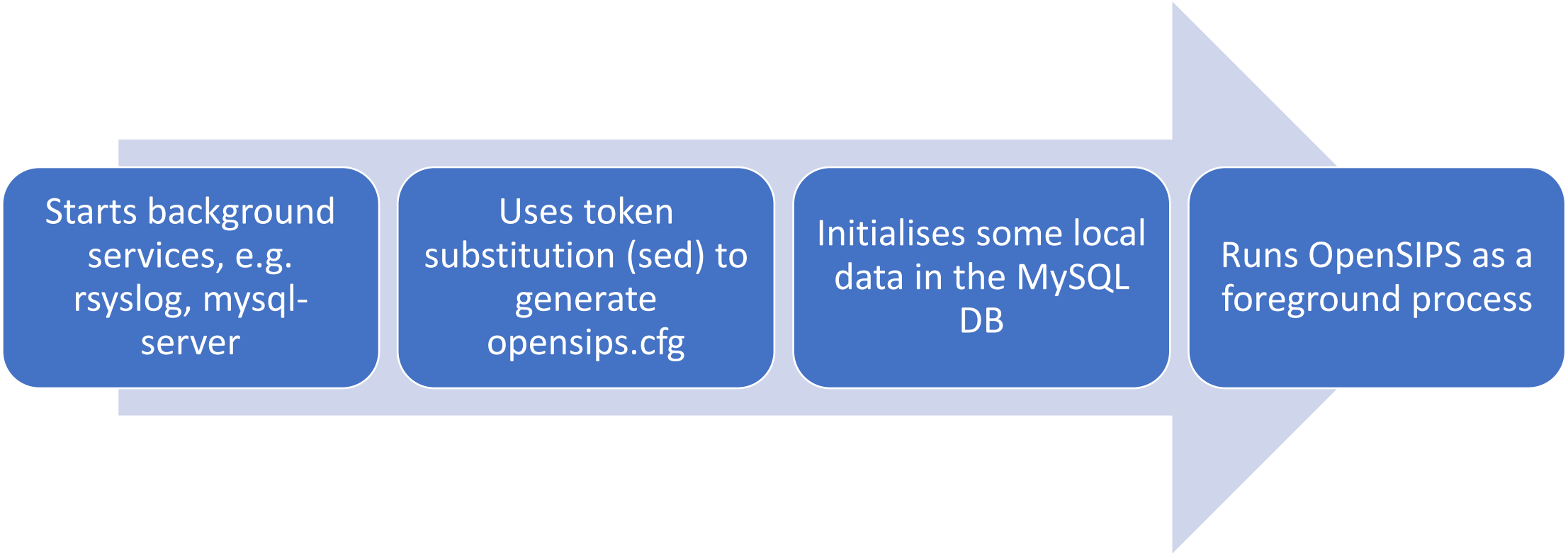
Docker Containers as Microservices

In Practice:

A Container is **primarily** used for one application	Last action during start-up runs a bash script
➢ e.g. I included a local instance of MySQL Server!	• The last action of the bash script runs the OpenSIPS app • Is this Cheating??

Docker Containers as Microservices

In Practice: What my bash script does



Example start-up bash script file

```
#!/bin/bash
cat /etc/opensips/opensips.cfg.template | sed "s/%HOST_IP_ADDR%/${HOST_IP}/g" |
sed "s/%CLUSTER_NODE_ID%/${NODE_ID}/g" > /etc/opensips/opensips.cfg

service rsyslog start
service mysql start
sleep 2

if [ -e /etc/opensips/dbinit.sql ] ; then
  mysql --user=root --password=mypasswrd opensips < /etc/opensips/dbinit.sql
  rm -f /etc/opensips/dbinit.sql
fi
/sbin/opensips -f /etc/opensips/opensips.cfg -F -m 640 -M 5
```

Docker-related Issues

Docker Containers as Scalable Cluster Nodes

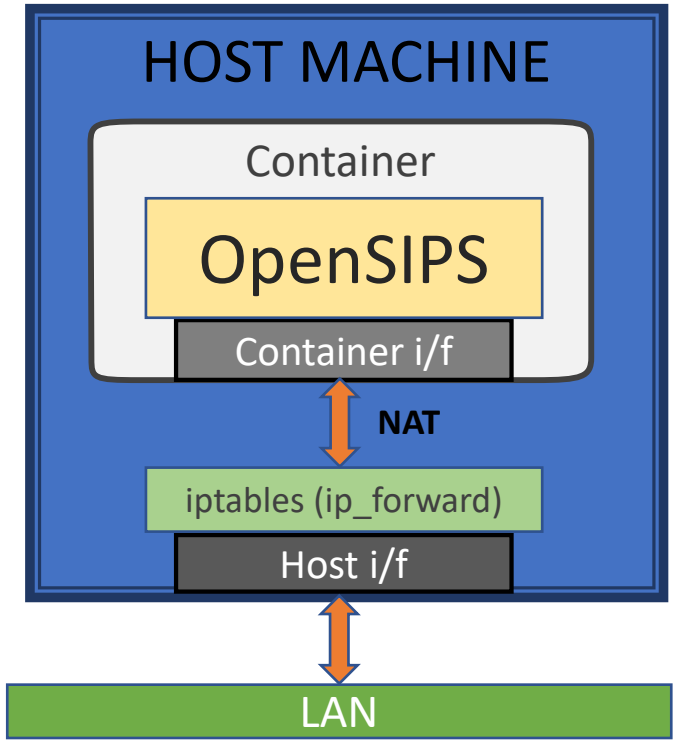
Idealised Scenario

- Every container is identical
- Automatic neighbour discovery

OpenSIPS clusterer nodes

- Unique Node ID
- Special “seed” node
- Minimum of one static node address

Containerisation and Docker networking



- OpenSIPS is effectively behind NAT
- Docker re-organises iptables rules
- Fail2ban jails must use FORWARD chain
- Where to run Certbot & store TLS Certificates?
- Errors in opensips.cfg are hard to diagnose
- Problem diagnosis is generally more difficult
- Log files – redirect to host or syslog server

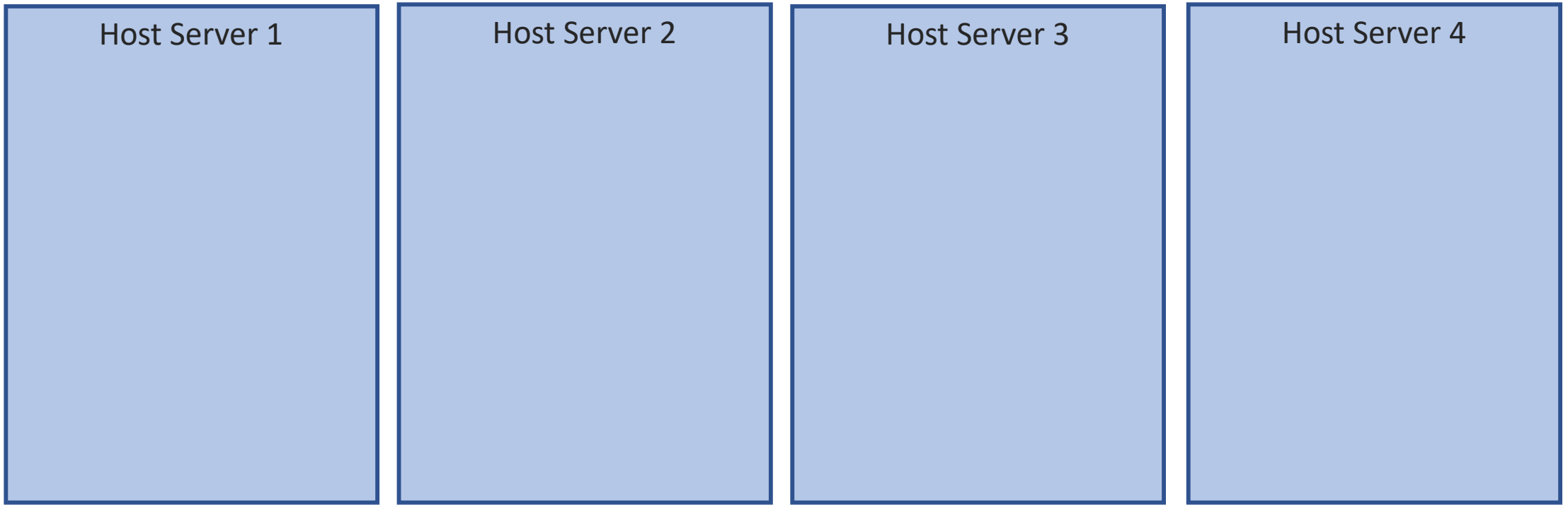
Running under Kubernetes

Work in progress

The role of Kubernetes

Define the physical resources: Nodes

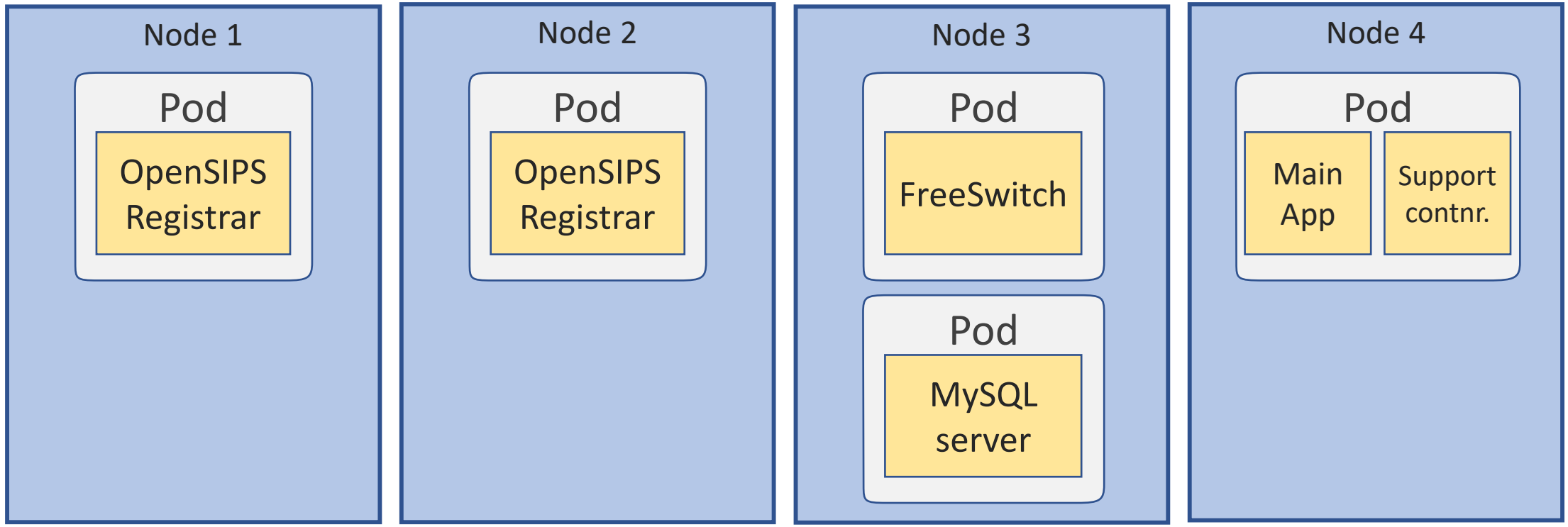
Nodes are assigned roles: Masters and Workers



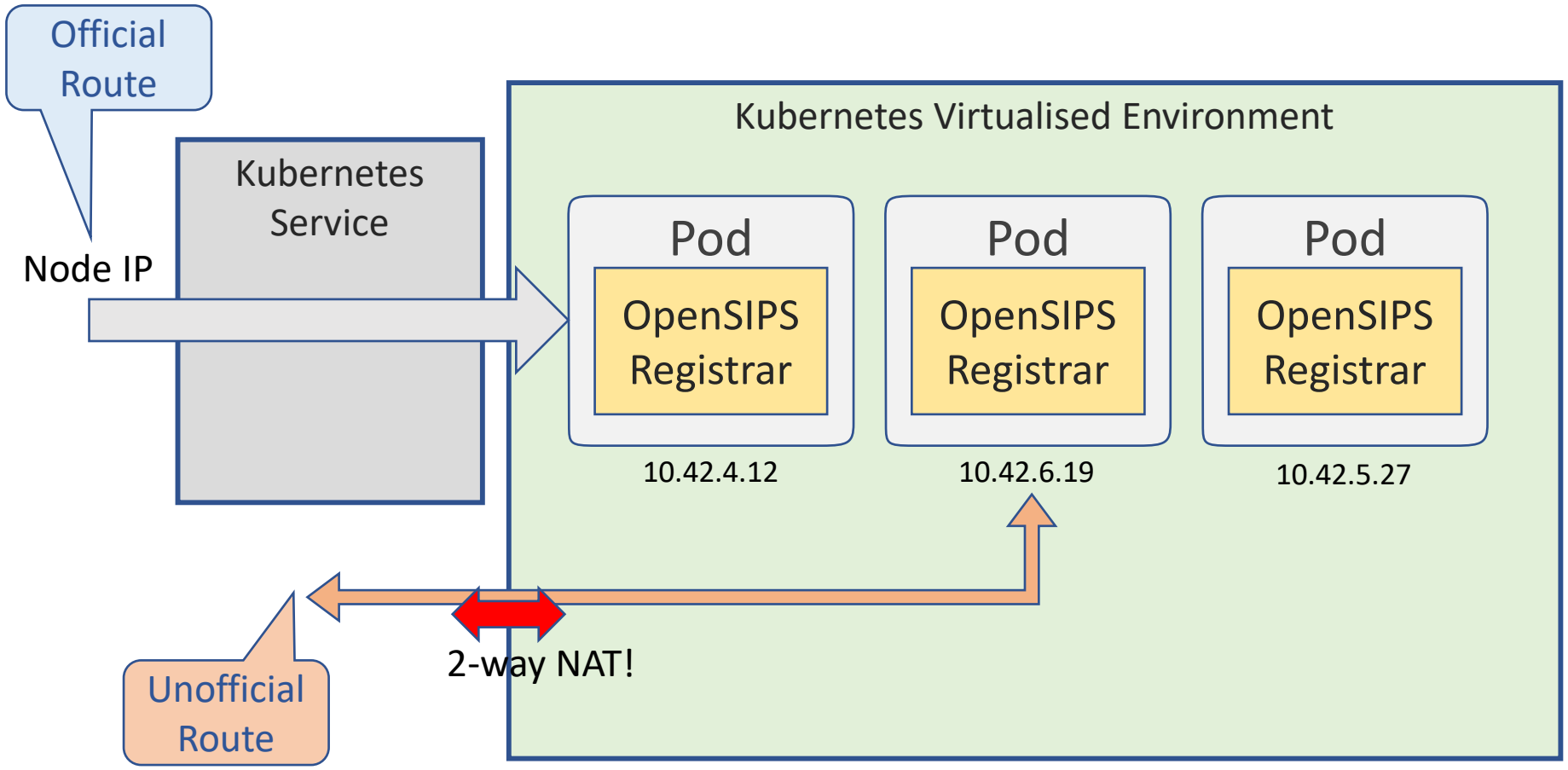
Define Apps: Containers running inside Pods

Pod name
How many replicas
Container source image

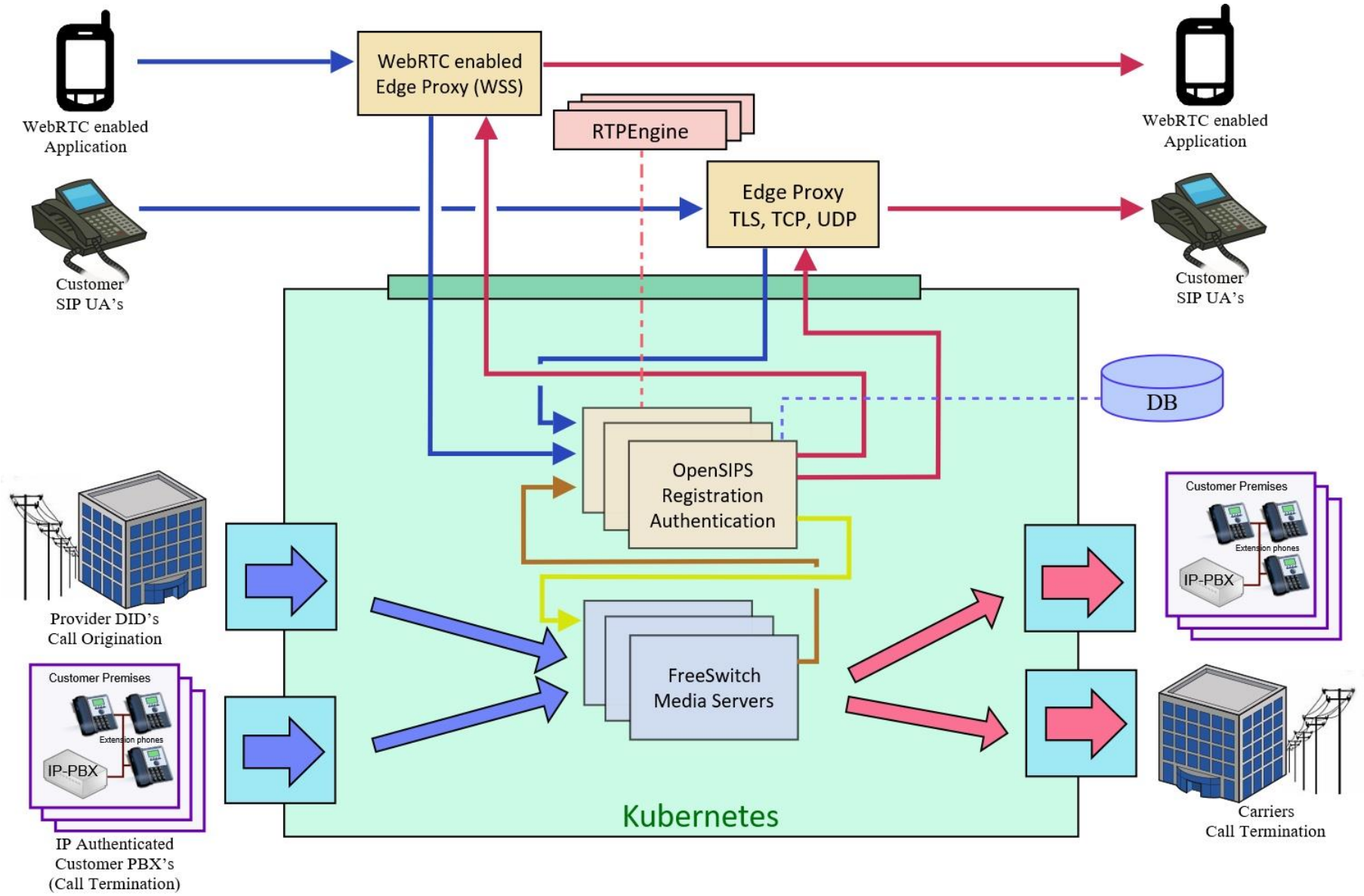
Ports to expose
Resource limits: Mem, CPU
Metadata



Kubernetes Services and Networking



Where Things Stand Today



Thank You.

…and goodbye

John Quick

Smartvox Limited

Retiring this year 2019

Any Questions?

John Quick

Smartvox Limited