VoIP & Kubernetes
(scalabilità) verso l'infinito … e oltre !

WHOLESALE WINERY FT Tour

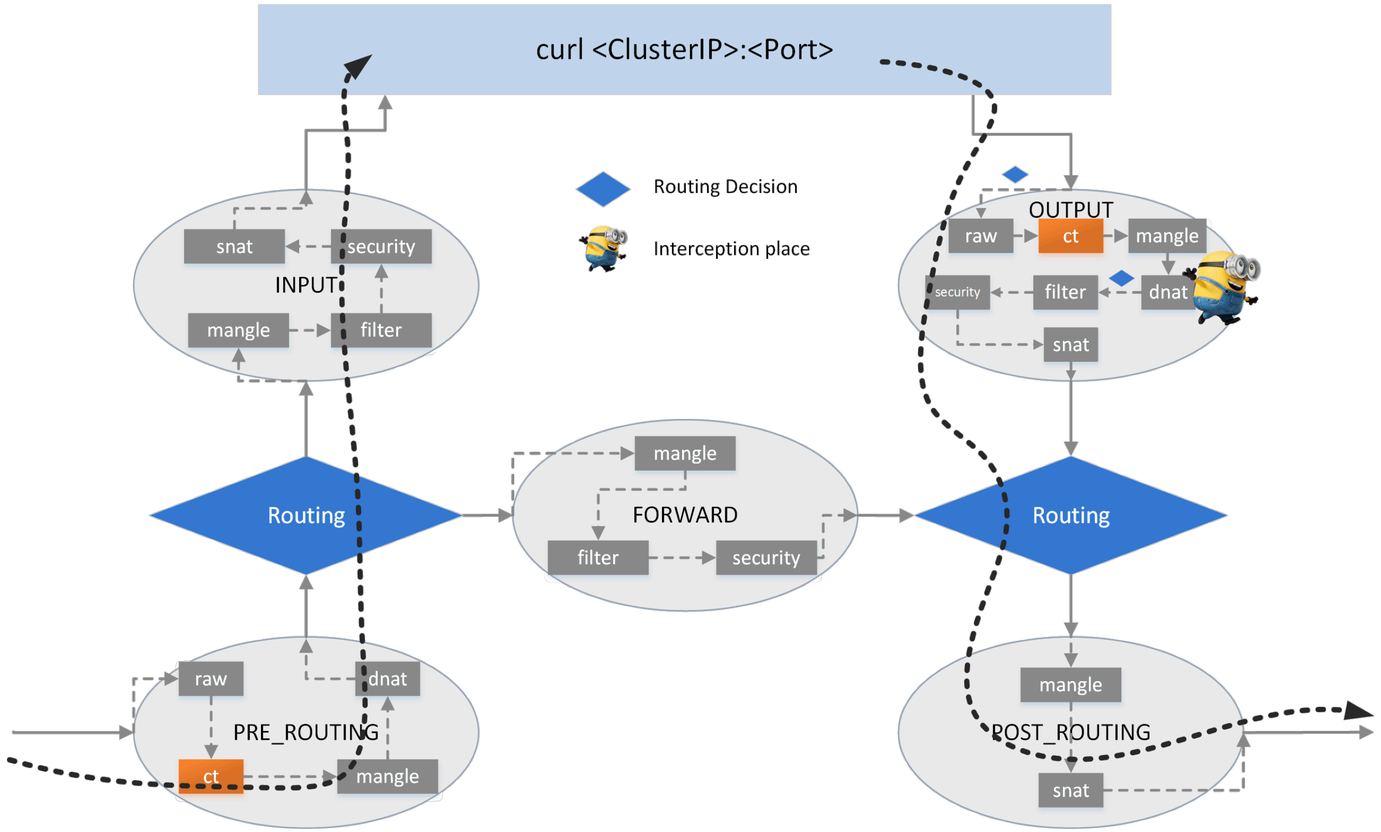
Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

Come può un'architettura di questo genere ...

kubernetes

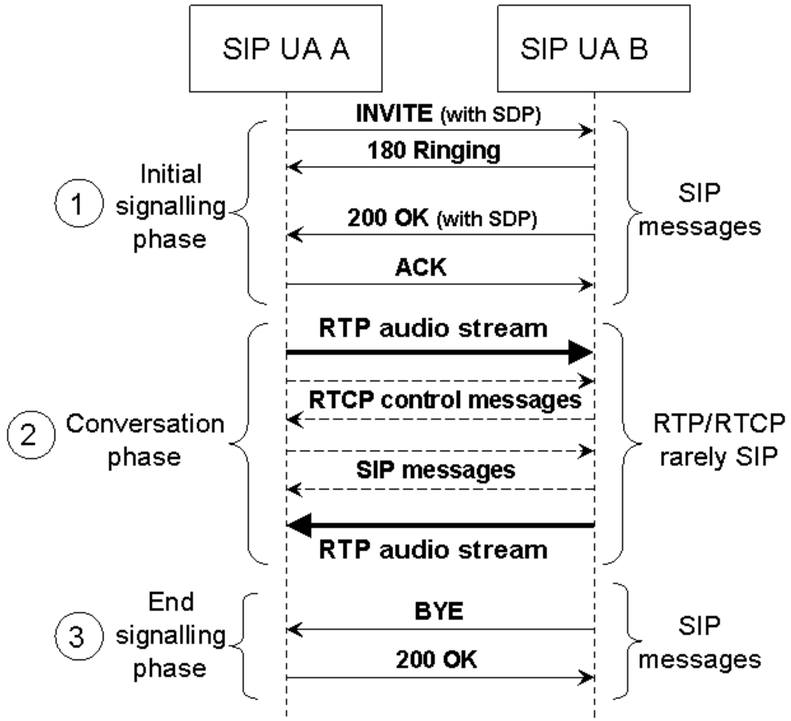


Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

… non fare a cazzotti con un protocollo di questo tipo ?

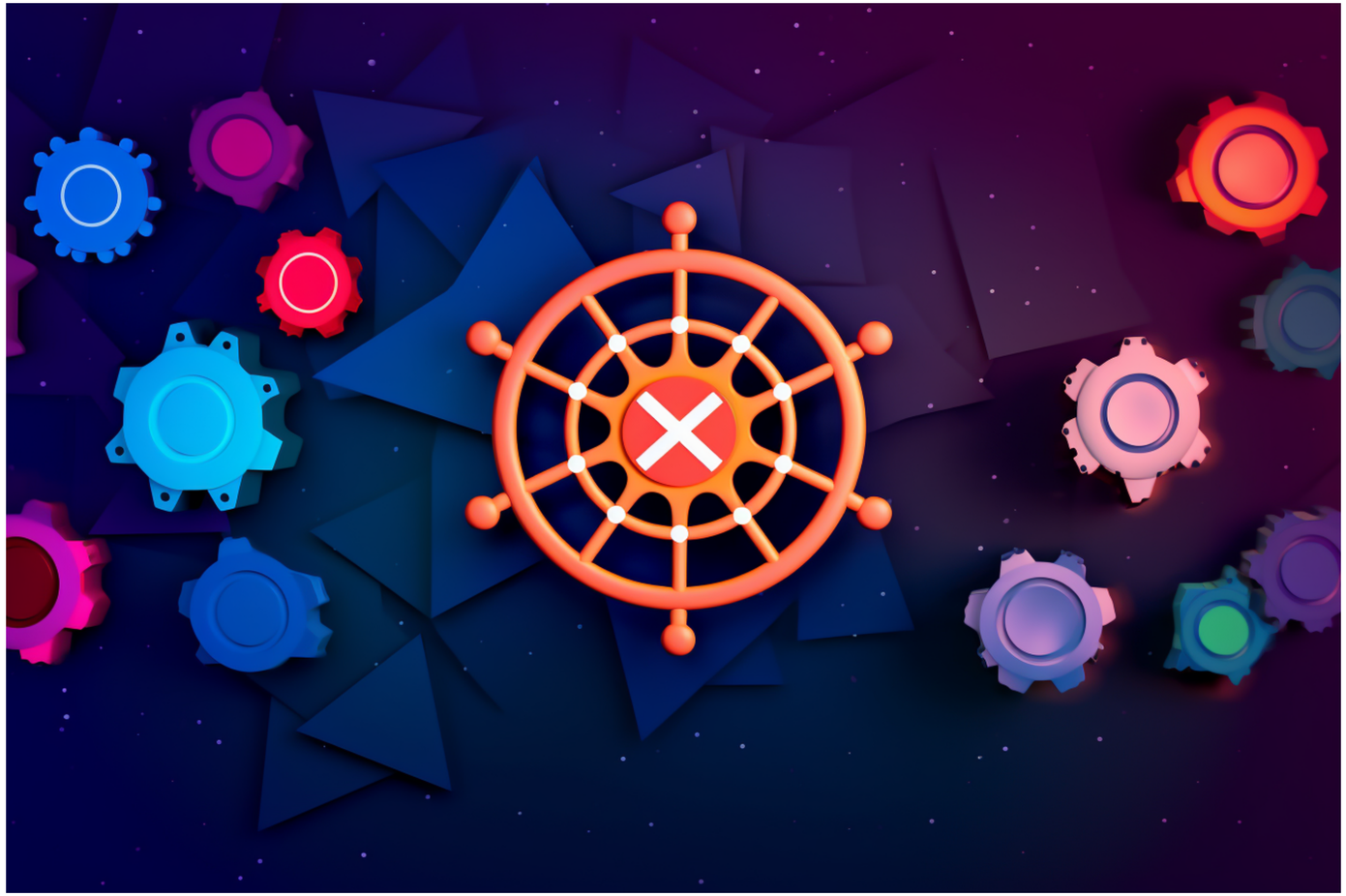
VoIP (SIP / RTP)



- NAT unfriendly
- Prevalentemente UDP
- Sensibile a MTU, latenza e Jitter

E come può il VoIP essere “stateless”

Per garantire un’eccellente disponibilità e scalabilità di servizio ?



Quali sono le relazioni tra:

Docker, Kubernetes, Kamailio, Asterisk, Consul e OSPF

WHOLESALE WINERY FT Tour

Paolo Visintin

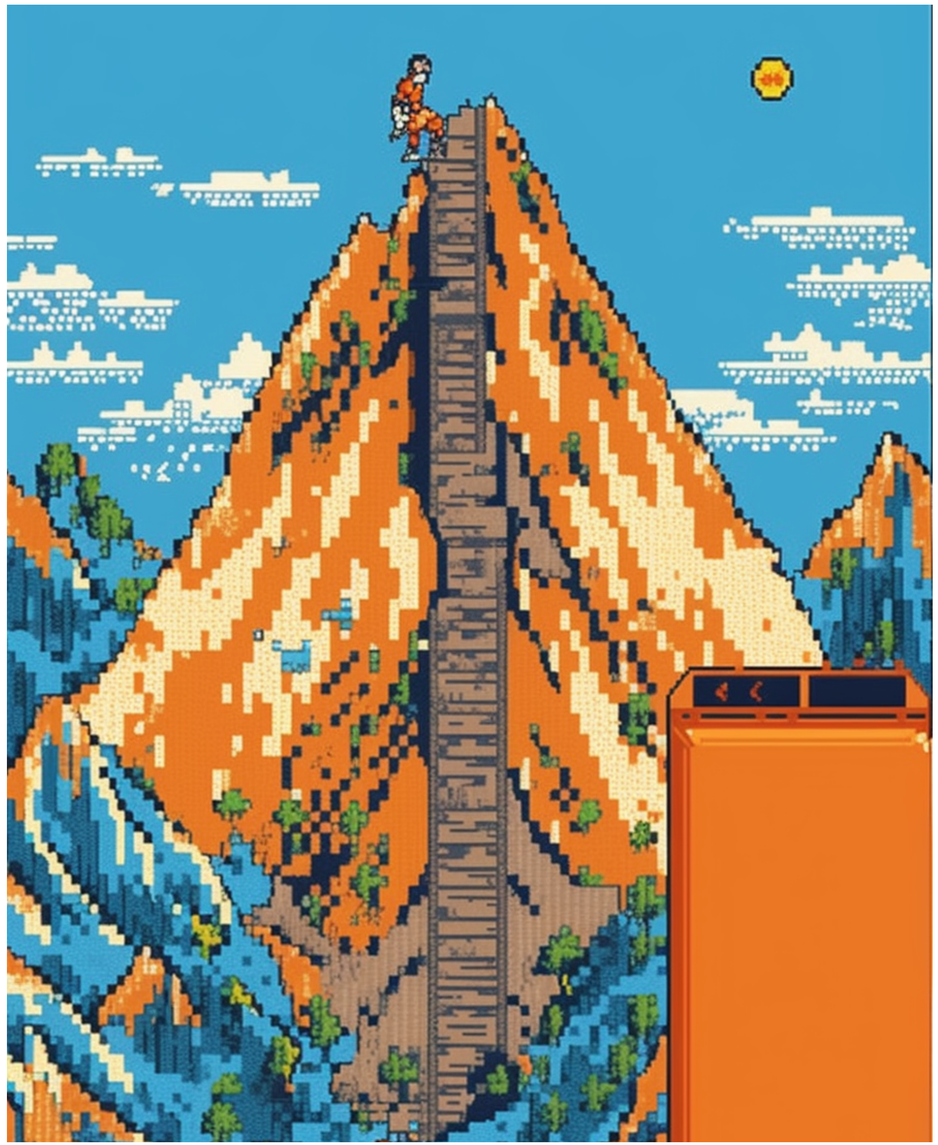
VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

Dove mi porterà questo viaggio epico ?

Posso dar vita ad un’architettura in grado di :

- **Scalare** senza limiti
- Farlo velocemente e **automaticamente**
- **Distribuirsi** su diverse architetture
- Non avere alcun vendor **lock-in**
- Garantire la **business continuity**



Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

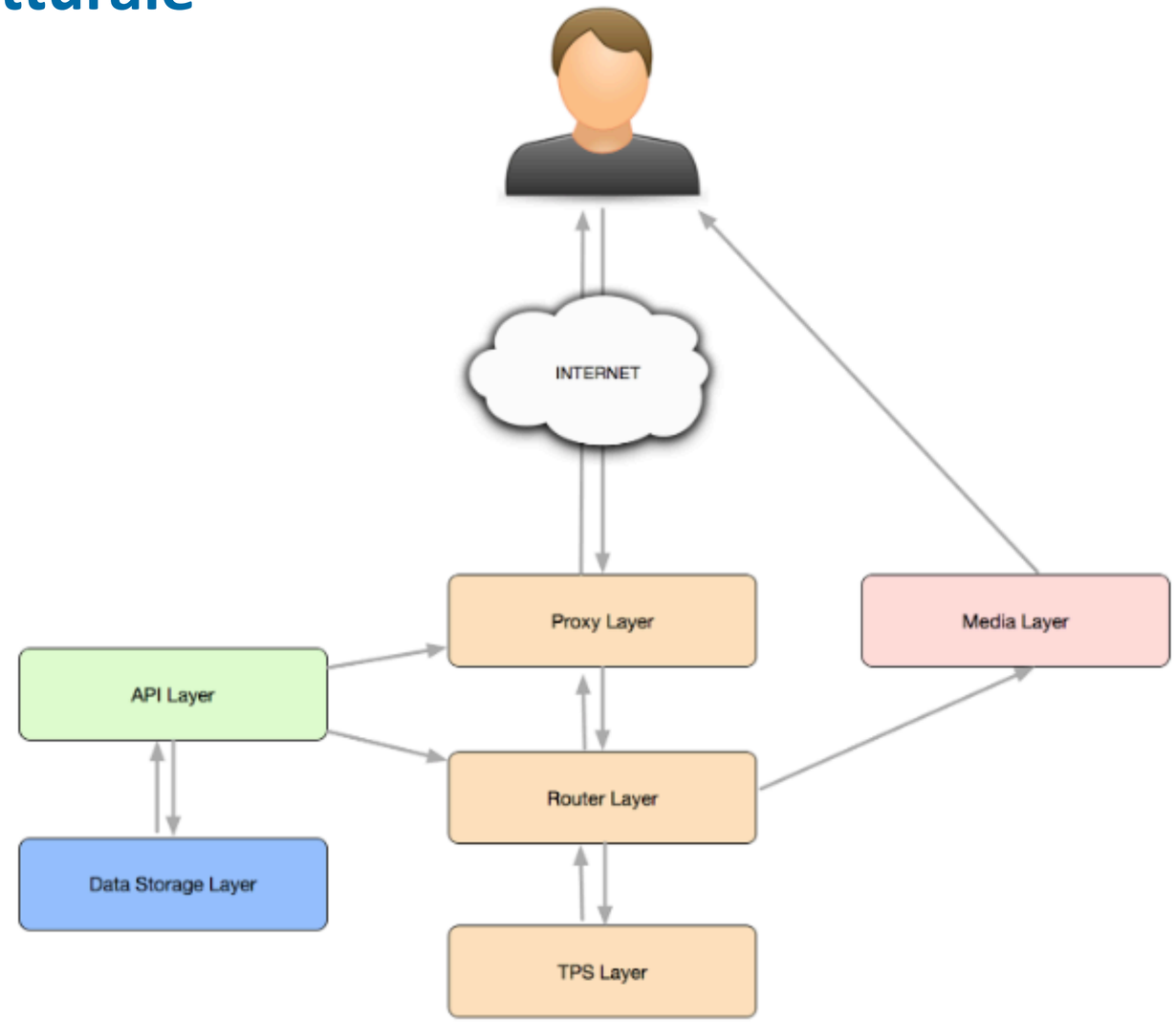
Ingredienti per la preparazione della ricetta

- **Docker** come engine per i container
- **Kubernetes** come sistema di orchestrazione dei servizi Docker
- **Kamailio** - il SIP router / proxy open source iper performante e versatile
- **Asterisk** - noto sistema open source per la gestione applicativa di servizi VoIP
- **un servizio HTTP API** per gestire la business logic
- **Consul** per gestire la service Discovery e la parte di DNS SRV

DISCLAIMER

Data la quantità di argomenti che potremmo trattare e il poco tempo a disposizione, volutamente tralascerò alcune parti essenziali di un sistema telefonico come, ad esempio, l'interconnessione con la rete pubblica o la parte di topology hiding o ancora tutta la parte di dastore; se avete piacere di approfondire questi come altri temi, confrontiamoci pure a voce a margine dello speech!

Concept Architetturale



Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

Quali sono le sfide da cogliere ?

- **PAIN POINT 1** - Networking
- **PAIN POINT 2** - Business continuity sotto controllo
- **PAIN POINT 3** - come gestire in modo smart la business logic
- **PAIN POINT 4** - Asterisk non è un tutto fare, anche se può fare tutto
- **PAIN POINT 5** - Media Relay - alte performance grazie al Kernel (di chi ?)
- **PAIN POINT 6** - service discovery senza scrivere codice

PAIN POINT 1 - Networking

Kubernetes & Docker : DNAT + SNAT + overlay

VoIP (SIP & RTP) : TCP/UDP nat unfriendly, MTU issues, latency first

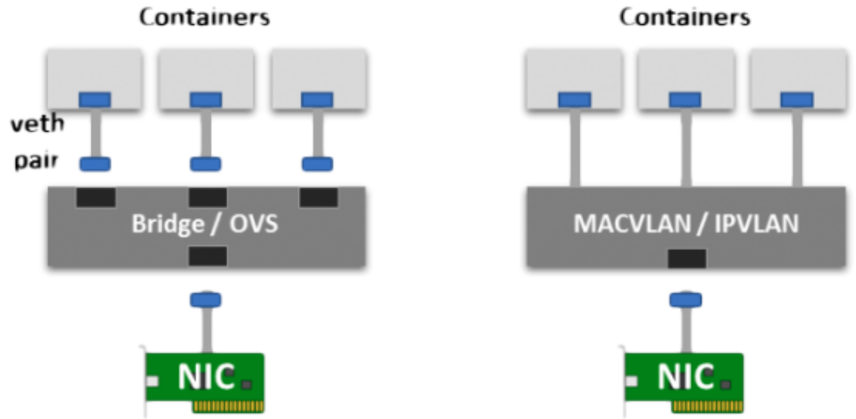
PAIN POINT 1 - Networking

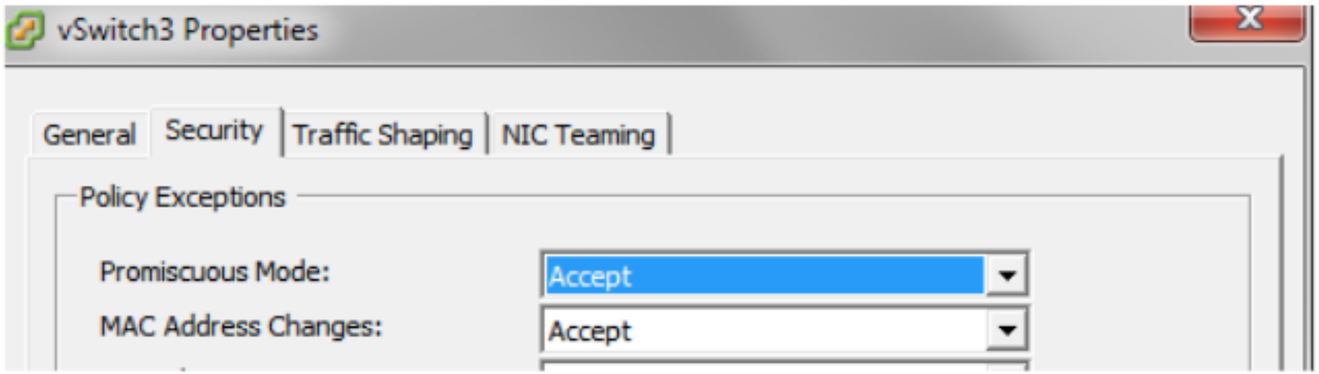
Kubernetes & Docker : DNAT + SNAT + overlay

VoIP (SIP & RTP) : TCP/UDP nat unfriendly, MTU issues, latency first

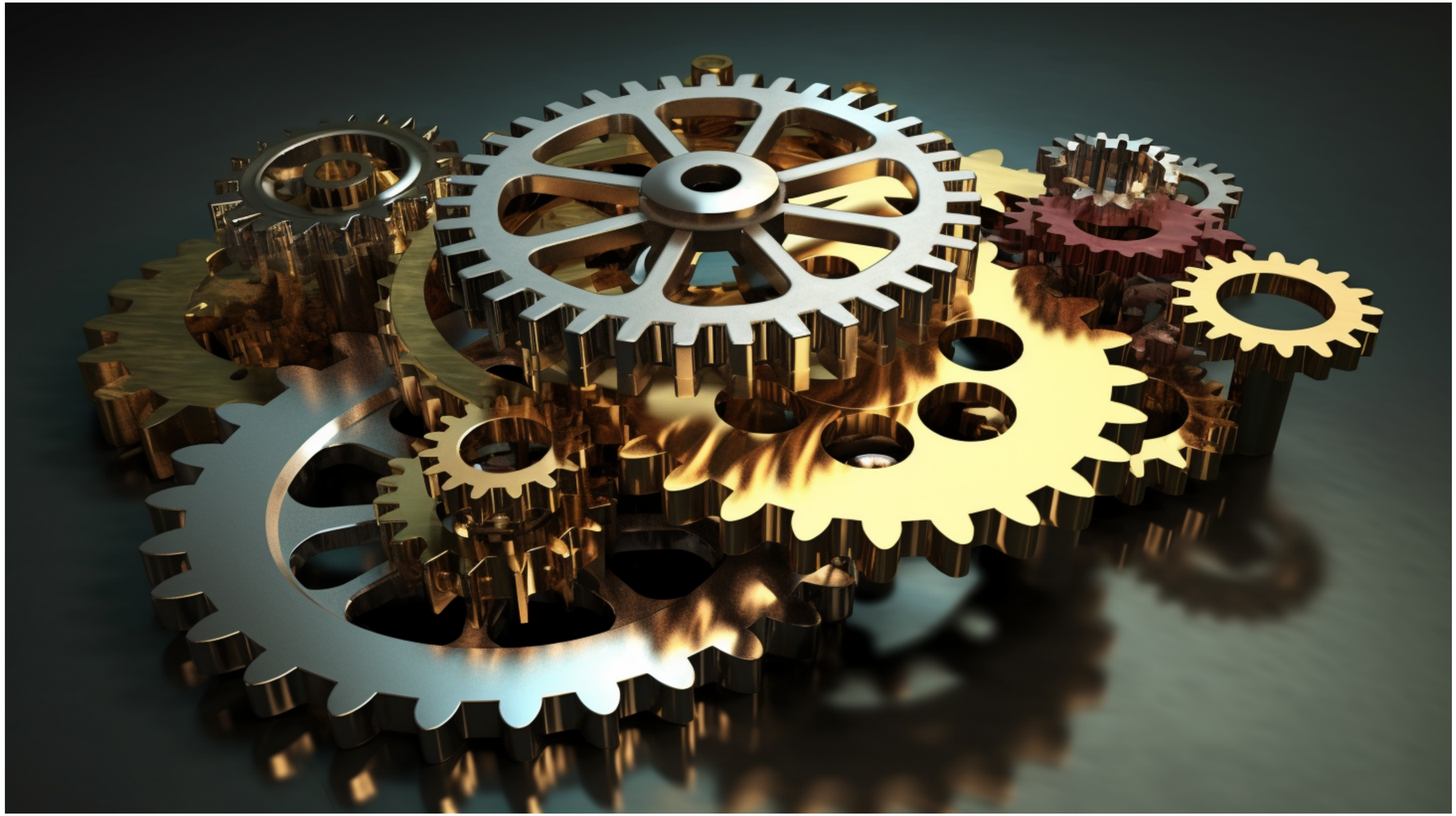
Soluzione 1	Soluzione 2
Proxy Layer come Ingress controller in modalità “network host” Tutti gli altri componenti che usano la CNI standard di Kubernetes **PRO**: compatibile con installazioni Kubernetes “standard” e distribuite **CONTRO**: meno flessibile, più overhead nella pila protocollare per incapsulamenti	utilizzo di “multi-pod network” come Multus CNI per assegnare più indirizzi IP ai POD e MACVLAN / IPVLAN come network per i servizi voce **PRO**: gestisco un L2 dentro ai container, segmento la rete per diversi servizi **CONTRO**: devo personalizzare parte dello stack di rete di Kubernetes, rendo il sistema più complesso; devo permettere il passaggio di mac address diversi sulla scheda di rete di ogni nodo con potenziali side-effect di pacchetti duplicati su alcuni hypervisor come VMWare

PAIN POINT 1 - Networking





PAIN POINT 2 - Business continuity sotto controllo



WHOLESALE WINERY FT Tour

Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

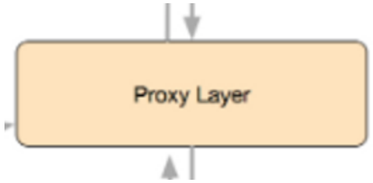
PAIN POINT 2 - Business continuity sotto controllo

Qual'è l'area più sensibile ?
Il punto di contatto tra il sistema e l'utente

Rendiamolo più leggero e performante possibile allora!

- Meno funzionalità
- Meno necessità di intervenire
- Meno richiesta di risorse

.. ecco che nasce il "proxy layer"



PAIN POINT 2 - Business continuity sotto controllo

300 CPS

50.000 concurrent calls

PAIN POINT 2 - Business continuity sotto controllo

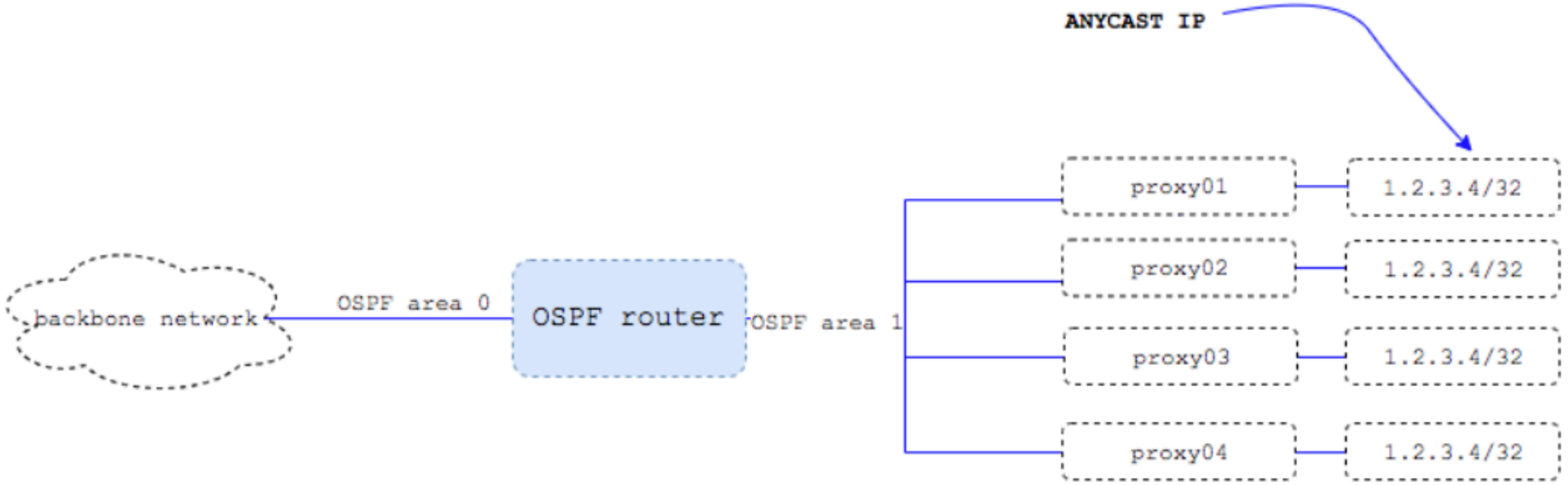
Come gestire il failover ?

DNS con TTL molto basso ? :

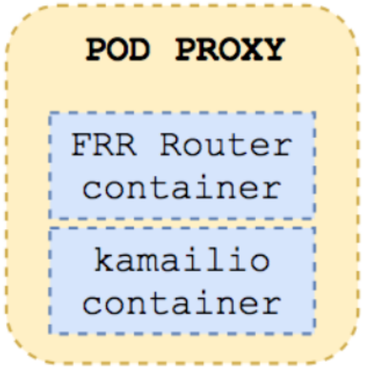
- Problema della cache lato cliente (device / router, applicazioni)
- Stiamo delegando al cliente il controllo

PAIN POINT 2 - Business continuity sotto controllo

OSPF con Anycast !



PAIN POINT 2 - Business continuity sotto controllo



```
FRR PROXY
...
interface eth0
 description *** OSPF interface ***
 ip ospf area 1
 ip ospf dead-interval minimal hello-multiplier 5

router ospf
 redistribute static
 passive-interface default
 no passive-interface eth0

ip route <proxy.ip>/32 Null0
ip route 0.0.0.0/0 <FRR_COREIP>
```

PAIN POINT 2 - Business continuity sotto controllo

Dopo aver stabilito il punto di contatto con il cliente, ho il pieno controllo nel gestire il traffico verso la mia "rete interna" come desidero, senza dover informare alcuno.

Ciò mi concede un notevole livello di:

- **libertà** negli aggiornamenti
- **scalabilità** dei servizi
- **flessibilità** nell'architettura

PAIN POINT 3 - come gestire in modo smart la business logic

+

Pensavamo che integrare in Kamailio il linguaggio Python per sviluppare la business logic, ci desse i

super poteri

PAIN POINT 3 - come gestire in modo smart la business logic

WHOLESALE WINERY FT Tour

Paolo Visintin

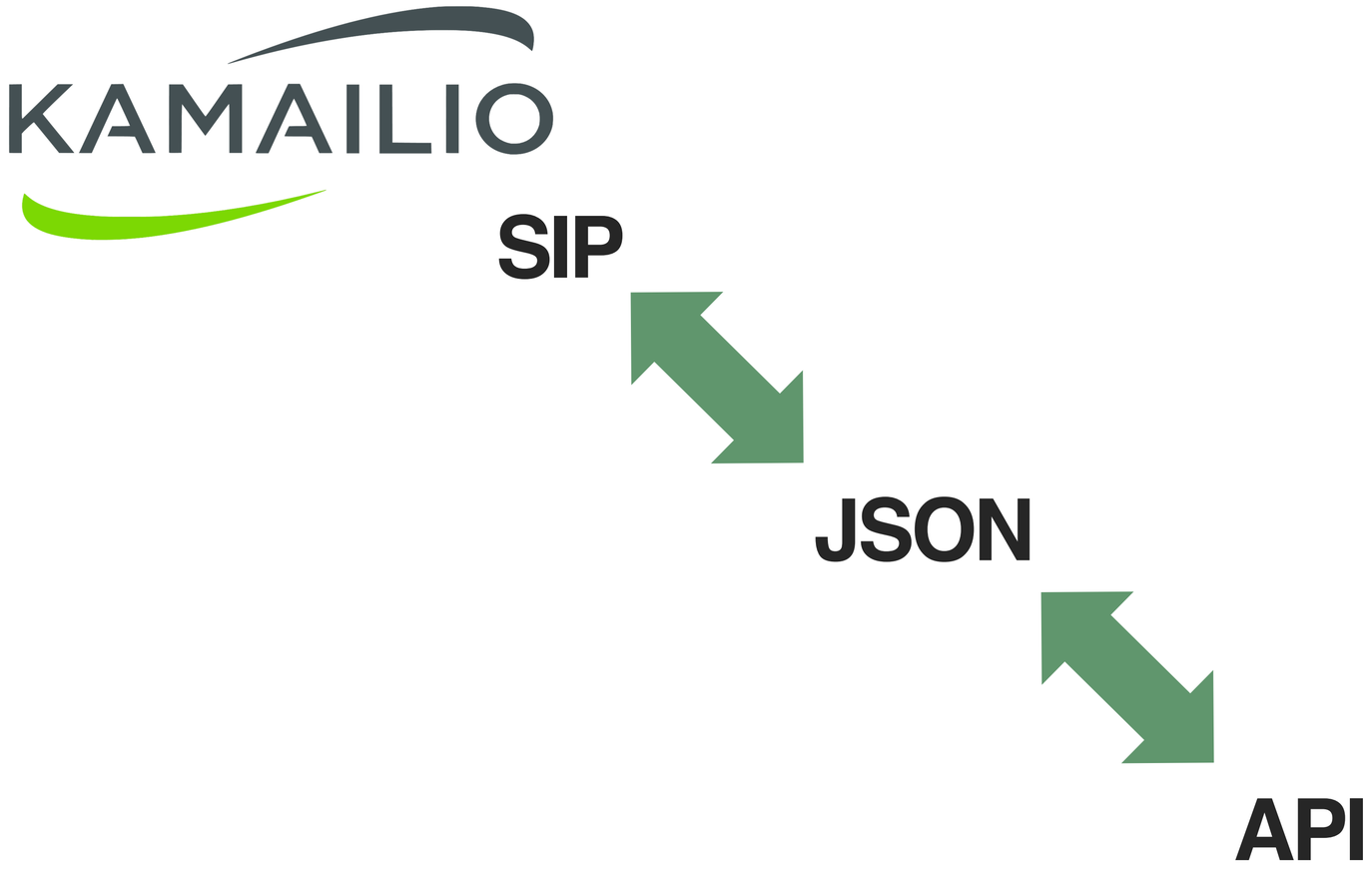
VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

PAIN POINT 3 - come gestire in modo smart la business logic

Questo invece è stato il risultato della collaborazione tra i Dev e i VoIP guys

PAIN POINT 3 - come gestire in modo smart la business logic



PAIN POINT 3 - come gestire in modo smart la business logic

WHOLESALE FT WINERY Tour

Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

PAIN POINT 4 - Asterisk non è un tutto fare, anche se può fare tutto

PAIN POINT 4 - Asterisk non è un tutto fare, anche se può fare tutto

Kamailio è estremamente performante MA non è:

- Un B2BUA
- Un Application Server

Asterisk, dal canto suo, è molto bravo a fare tante cose, ma non è altrettanto **performante!**

PAIN POINT 4 - Asterisk non è un tutto fare, anche se può fare tutto

Per questo motivo abbiamo deciso di utilizzare Asterisk :

- on demand (cioè solo quando serve)

- esclusivamente come application layer (voicemail, conferenze, IVR / Playback, etc)
- stateless e agnostico

PAIN POINT 4 - Asterisk non è un tutto fare, anche se può fare tutto

Come comunicare con Asterisk integrando le istruzioni logiche nel pacchetto SIP !

```
INVITE …
...
X-Asterisk-Action: Playback
X-Asterisk-Sound: mysoundfile
...
```

```
asterisk extensions.conf
[default]
exten => _X.,1,NoOP(:: dispatching requests ::)
exten => _X.,n,GotoIf($["${SIP_HEADER(X-Asterisk-Action)}" = "Playback"]?playback)
exten => _X.,n,GotoIf($["${SIP_HEADER(X-Asterisk-Action)}" = "Voicemail"]?voicemail)
```

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

Il VoIP è la somma di due protocolli, il protocollo **SIP**, e il protocollo **RTP**, che viene utilizzato per lo streaming media .

Il nostro “proxy” layer è deputato solamente alla gestione della parte SIP, quindi dobbiamo preoccuparci di gestire anche un “**media layer**” in grado di fare da bridge tra la rete esterna e la rete interna .

SIP

+

RTP

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

Il protocollo SIP in realtà è deputato anche alla negoziazione della parte media, infatti in alcuni pacchetti (come ad esempio l'INVITE) è presente solitamente una parte di **SDP** (Session Description Protocol) , ecco un esempio:

```
v=0
o=- 3174032640 1 IN IP4 77.239.128.234
s=-
t=0 0
m=audio 44294 RTP/AVP 8 18 101
c=IN IP4 77.239.128.234
a=maxptime:20
a=sqn:0
a=cdsc:1 image udptl t38
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=ptime:20
```

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

```
v=0
o=- 3174032640 1 IN IP4 77.239.128.234
s=-
t=0 0
m=audio 44294 RTP/AVP 8 18 101
c=IN IP4 77.239.128.234
a=maxptime:20
a=sqn:0
a=cdsc:1 image udptl t38
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=ptime:20
```

In questo pezzo di pacchetto **INVITE** vediamo che il sender specifica che vuole essere contattato sull'indirizzo IP 77.239.128.234 alla porta 44294 , accetta il T38 ed il payload è 8 18 101: Questi sono i formati di payload supportati per questo flusso audio.
Ogni numero corrisponde a un codec specifico o a un'informazione di controllo del flusso.
In questo caso, i numeri corrispondono a:

- 8: G.711 a-law
- 18: G.729
- 101: Telephone Event (RFC 2833), utilizzato per la trasmissione di eventi DTMF (Dual Tone Multi-Frequency)

Dobbiamo quindi manipolare la parte SDP per gestire correttamente il Media Bridging

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

sipwise/**rtpengine**

The Sipwise media proxy for Kamailio

Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

Una caratteristica fondamentale di RTPEngine, che lo rende un prodotto particolarmente performante è quella di lavorare in **kernel space**, taggando i pacchetti RTP e girandoli in una specifica kernel table.

```
iptables -I rtpengine -p udp -j RTPENGINE --id "$RTP_FORWARDING_TABLE”
```

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

Come faccio a caricare un modulo kernel in un container

che non ha il kernel ?

In realtà docker eredita il kernel del host !

PAIN POINT 5 - Media Relay - alte performance grazie al Kernel (di chi ?)

- Installo il modulo kernel xt_RTPENGINE sui worker e si fa in modo che carichi all'avvio

- Gestisco nel bootstrap del container RTP una allocazione intelligente delle "kernel tables"

- Eseguo il tagging per id dentro ogni singolo container

… ed ecco che abbiamo così implementato il media bridging, altamente performante (in kernel space), distribuito su cluster Kubernetes.

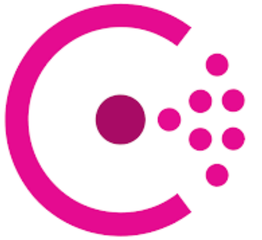
PAIN POINT 6 - service discovery senza scrivere codice

- A livello architetturale finalmente ci siamo
- Dinamicità e flessibilità di un sistema si misurano anche su come i componenti parlano tra loro
- La sovra-ingegnerizzazione è dietro l'angolo
- La sfida è rendere tutto più semplice e efficace

PAIN POINT 6 - service discovery senza scrivere codice

PAIN POINT 6 - service discovery senza scrivere codice

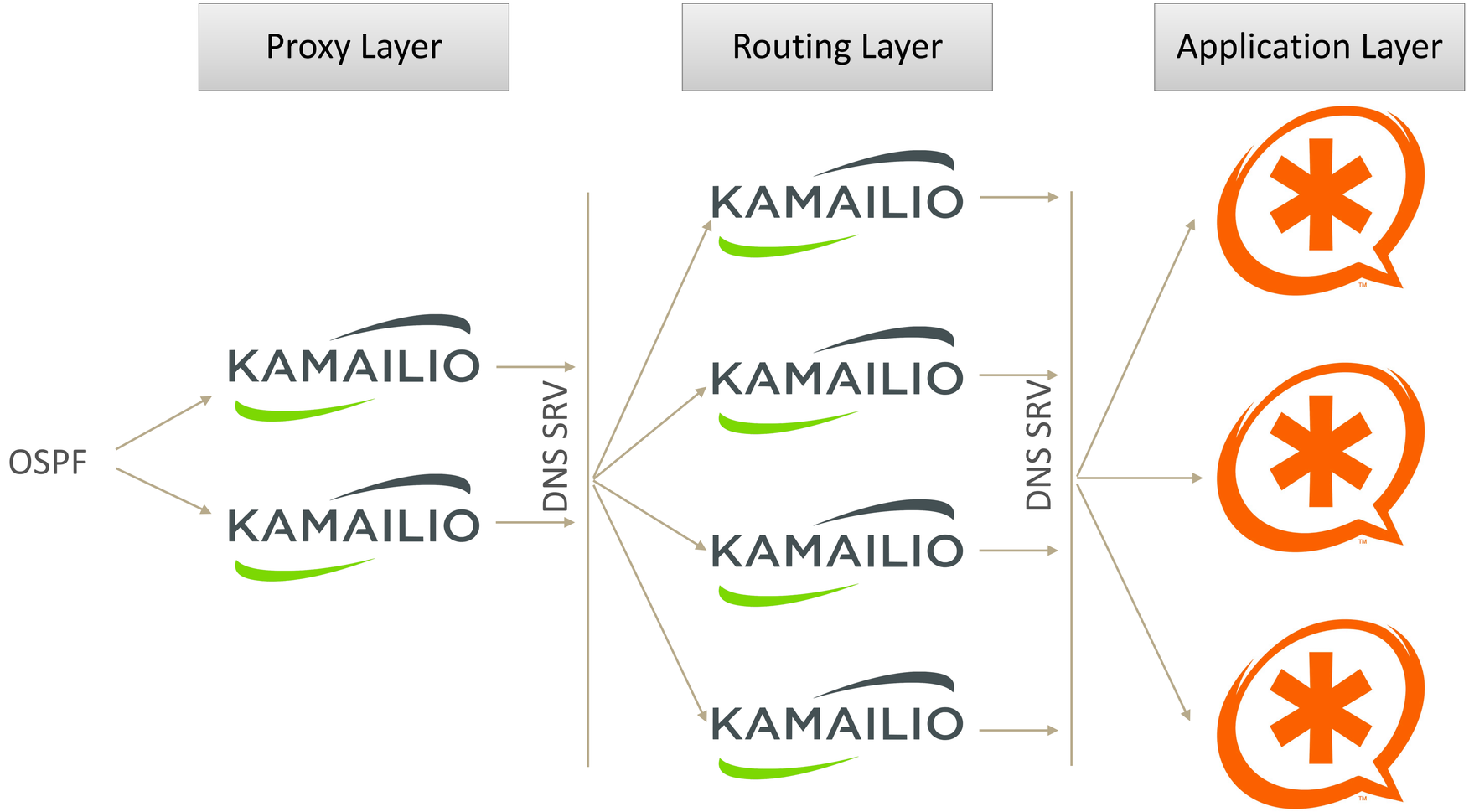
- Consul è un Service Discovery che posso utilizzare per ogni bootstrap di ogni servizio
- Consul espone le proprie informazioni anche come DNS Server
- Supporta la risoluzione dei record DNS SRV per poter fornire informazioni sui servizi registrati, tra cui indirizzi IP, porte e altre proprietà configurabili



PAIN POINT 6 - service discovery senza scrivere codice

- Kamailio permette di impostare come “destination” un DNS (e utilizza query SRV)
- Se trova più di un record fa load balancing in automatico

PAIN POINT 6 - service discovery senza scrivere codice



Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

La compagnia del pacchetto (SIP)

WHOLESALE WINERY FT Tour

Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

Nessun cluster Kubernetes è stato maltrattato per realizzare questa presentazione!

WHOLESALE WINERY FT Tour

Paolo Visintin

VoIP & Kubernetes - come realizzare architetture che scalano in automatico

ITNOG ON THE ROAD

Restiamo in contatto !

paolo.visintin@evoseed.io

https://www.linkedin.com/in/paolo-visintin/