evosip
the world outloud

Kamailio with Docker and Kubernetes

Scale in the right way

KWC 2018 - Berlin

about me...

My name is Paolo Visintin

I'm passionate about telecommunications and technology, I design, develop and maintain VoIP platforms based on Kamailio and Asterisk

I'm the CTO of an Italian telco operator providing internet and voice services in the Country

I decided to start a new project named **evosip**, a cloud PaaS VoIP infrastructure built on kubernetes

You can follow me:

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

http://evosip.cloud/

Agenda

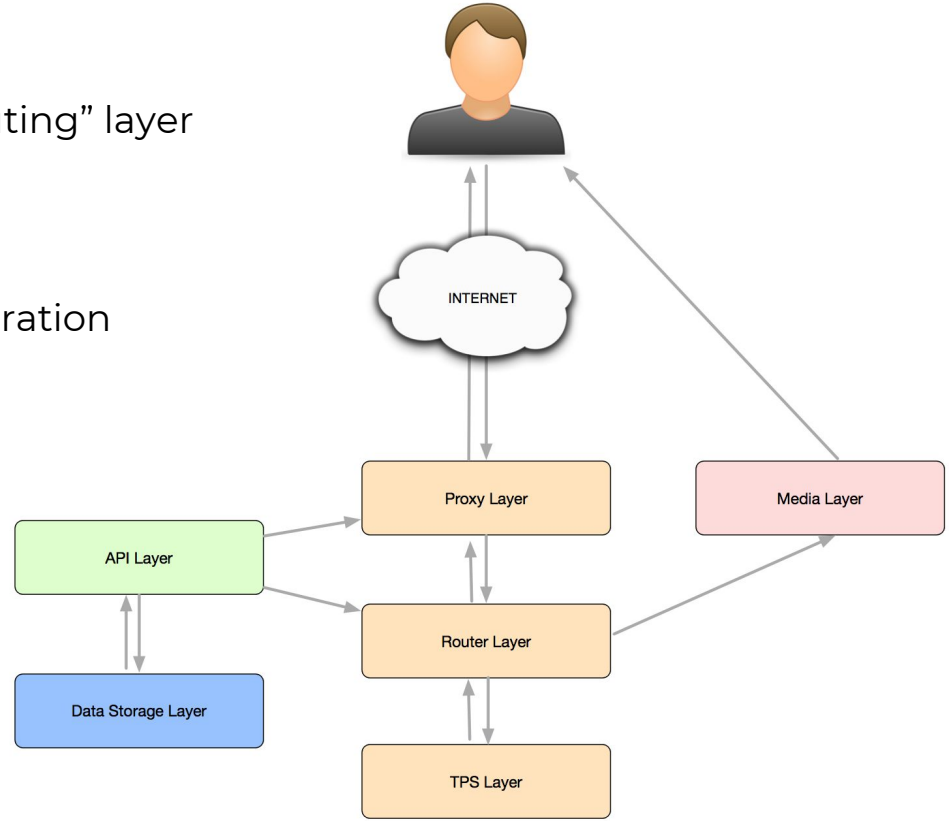
In this speech I'd like to tell you how we created a platform that:

- scales with no limits
- scales fast and automatically
- is distributed
- is QoE oriented
- has no vendor lock-in
- achieve business continuity

...a platform we named **evosip!**

evosip diagram

- Kamailio proxy and “routing” layer
- Asterisk as TPS
- RTPEngine
- Custom API SIP orchestration



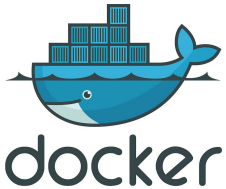
evosip ... from the beginning

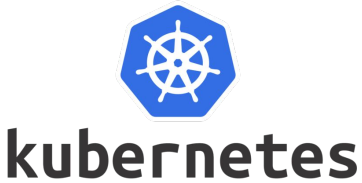
We were looking for a platform able to :

- create instances in a very fast way (couple of seconds)
- orchestrate instances between layers
- orchestrate network and policies
- works in a distributed environment

So we started to use containerisation and saw it was really fast and useful

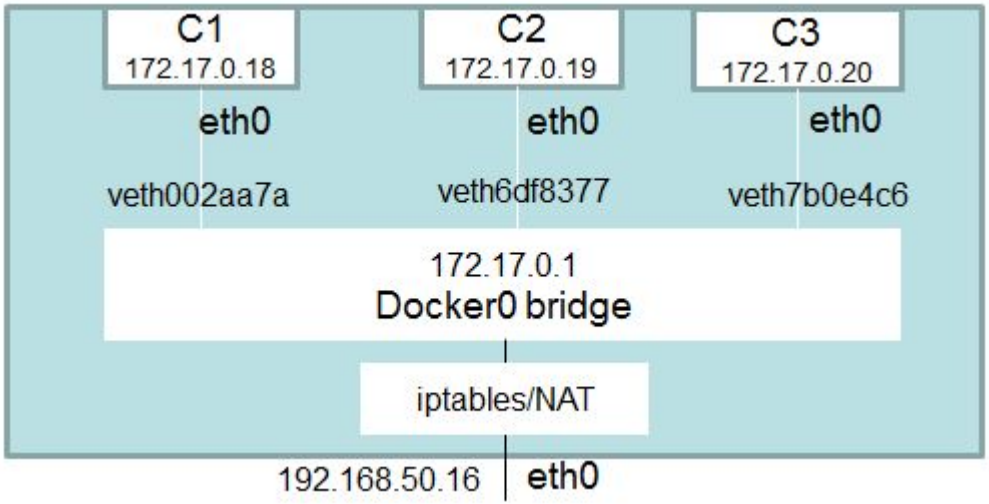
Using **Docker** for instance deployment and **Kubernetes** for orchestration (container, network and policies) we found an optimal stack to start this adventure and create a really fast and mutational SIP ecosystem based on Kamailio, Asterisk and RTPEngine





Containers … avoid network pain!

If you have used containers in a classical way probably you have noticed something terrible about dealing with networking and SIP

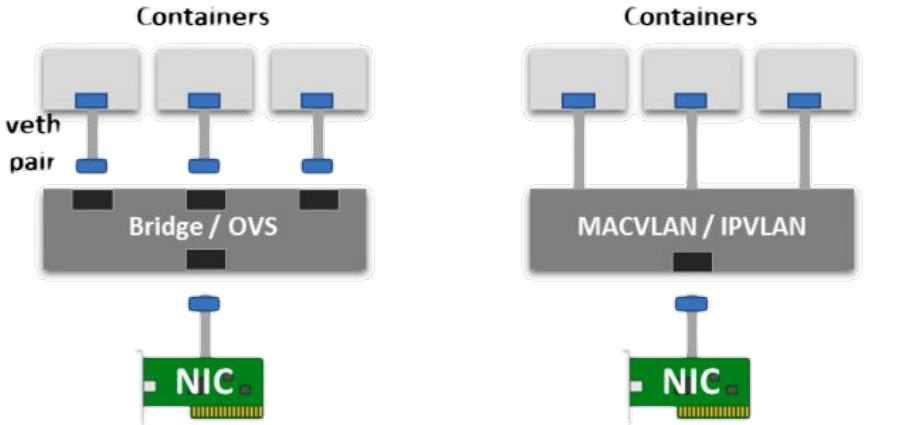


This diagram shows the classical use of networking with docker

Containers … avoid network pain!

So how could you use SIP in containers without doing NAT / bridging and avoid network issues and lack of performance ?

…with **macvlan** **!**

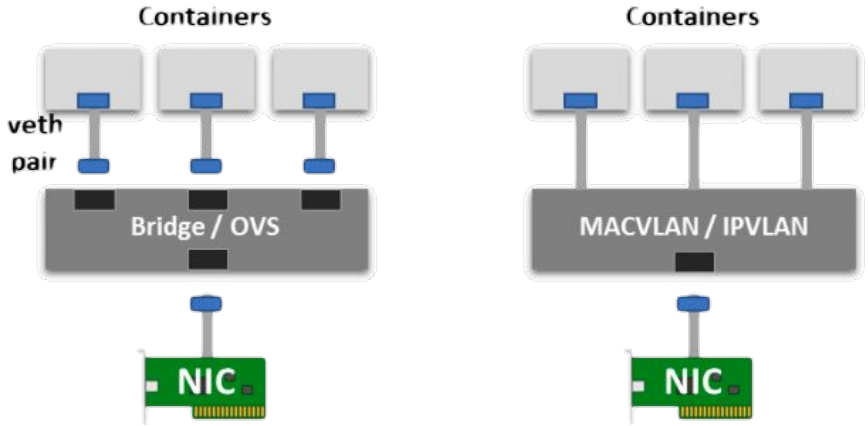


Differences between bridge networking and [mac/ip]vlan networking

Containers … avoid network pain (with macvlan)!

Thanks to macvlan every instance can have a direct public IP address with its own mac address

It uses less CPU and provides better throughput (almost like the physical interface)



macvlan associate to Linux Ethernet interface or sub-interface

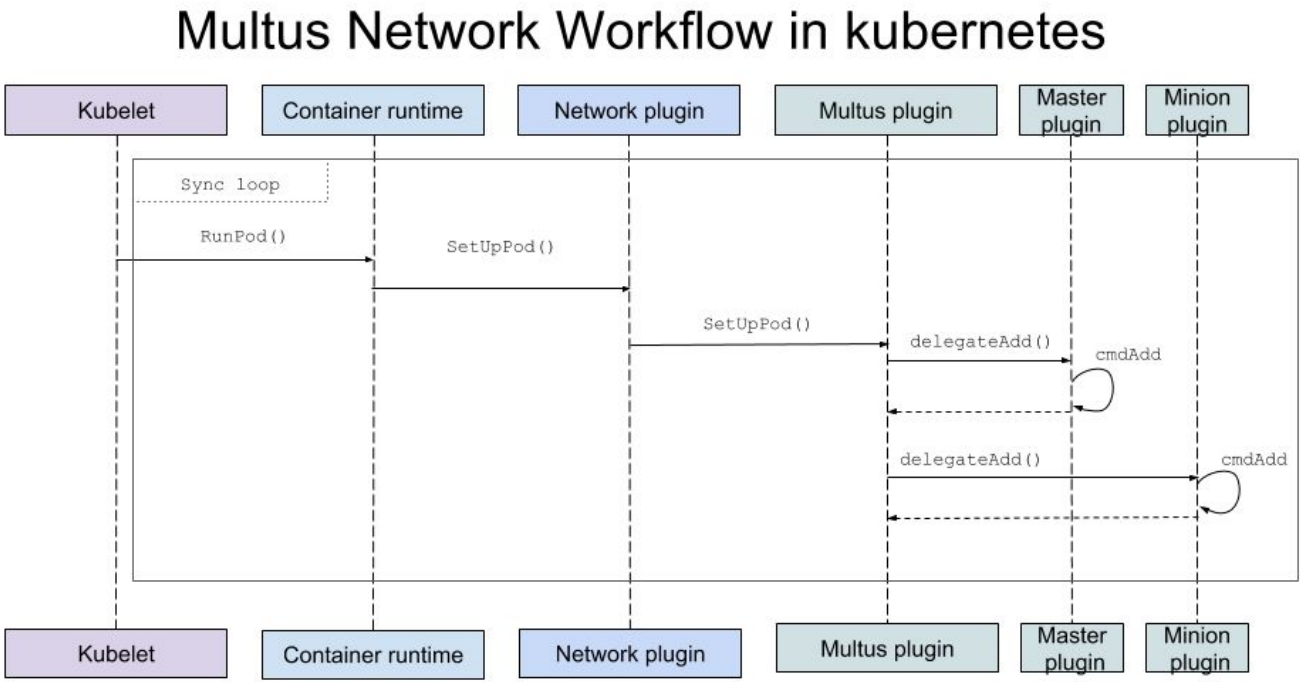
Containers with multiple networks

We also decided to separate

- data network (DB / API / etc)
- core service network (SIP / RTP)

But … if you know kubernetes, you also know that a POD (a group of one or more containers with shared storage/network) by default provides a single network interface

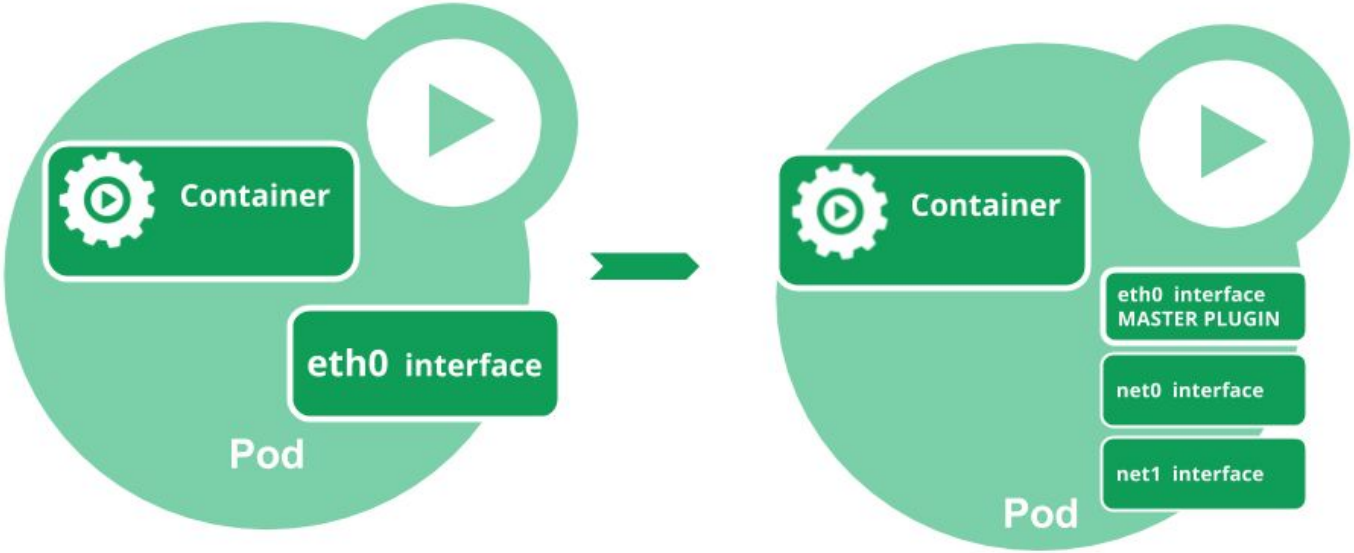
Containers with multiple network



with **multus** (https://github.com/Intel-Corp/multus-cni - a CNI plugin) you are able to do it!

evosip
the world outloud

Containers with multiple network



Source: Inspired from Vishnu kannan K8s Technical Deep Dive presentation

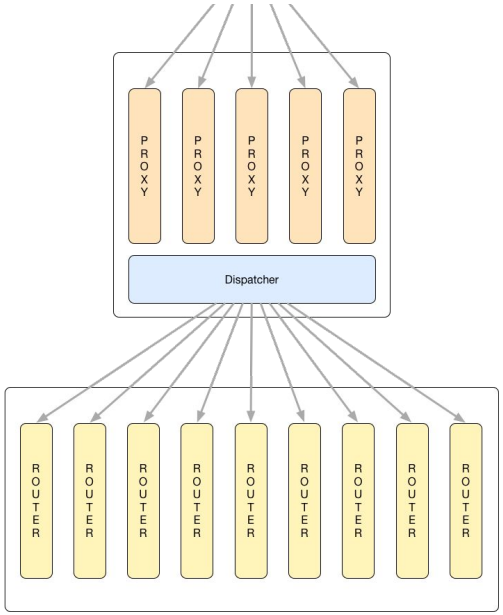
Kamailio ASAP (As Stateless As Possible)

Containerized services work and scale better with stateless applications

Then, **how to make kamailio *ASAP* ?**

We will now focus on :

- dispatcher
- authentication
- user location
- dialogs



Kamailio ASAP - cached dispatchers

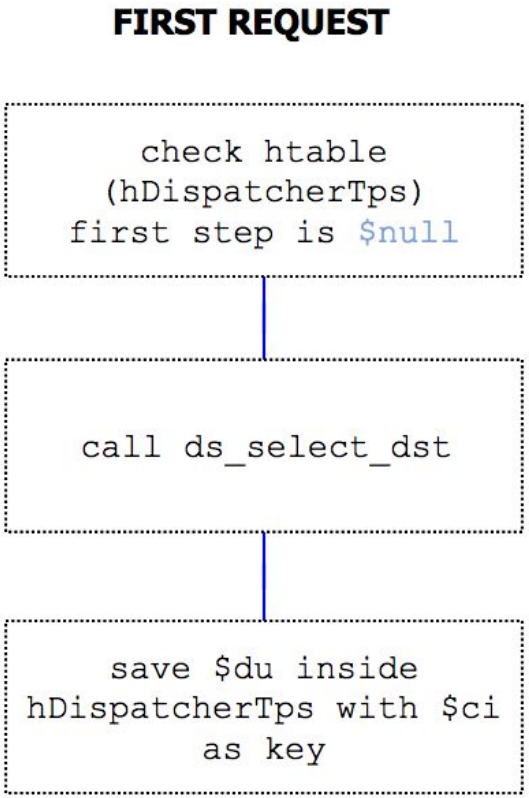
We use the dispatcher module to route request from proxy layer to router layer (both are kamailio auto-scalable instances)

As of the possibility of reloading the dispatcher module (e.g. autoscaling of routers) once used `ds_select_dst` we store the `$du` and callerid (`$ci`) in a hash table in order to re-use `$du` on the next request and maintain the path to the same endpoint for the call

We disabled the keepalive option in dispatcher to avoid pinging endpoints

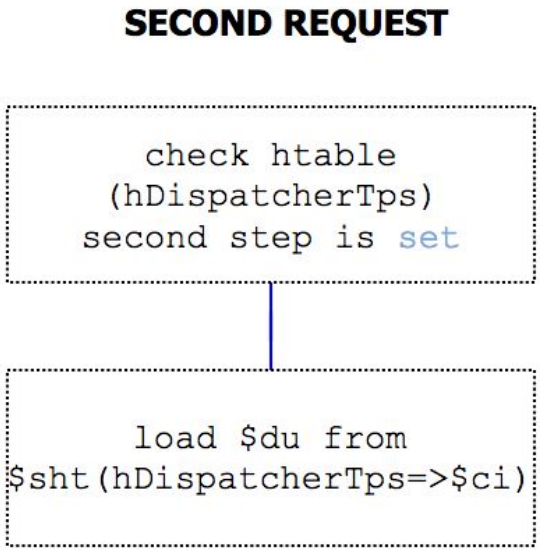
We set instead short timers to handle failure_route in case of tear-down on unreachability of endpoints

Kamailio ASAP - cached dispatchers




```
if ($sht(hDispatcherTps=>$ci)==$null) {
    # select tps from dispacher
    ds_select_dst("1","0");
    $sht(hDispatcherTps=>$ci) = $du;
  } else {
    $du = $sht(hDispatcherTps=>$ci);
  }
```

Kamailio ASAP - cached dispatchers




```
if ($sht(hDispatcherTps=>$ci)==$null) {
    # select tps from dispacher
    ds_select_dst("1","0");
    $sht(hDispatcherTps=>$ci) = $du;
  } else {
    $du = $sht(hDispatcherTps=>$ci);
  }
```

Kamailio ASAP - Auth module

How we implemented authentication in kamailio ?

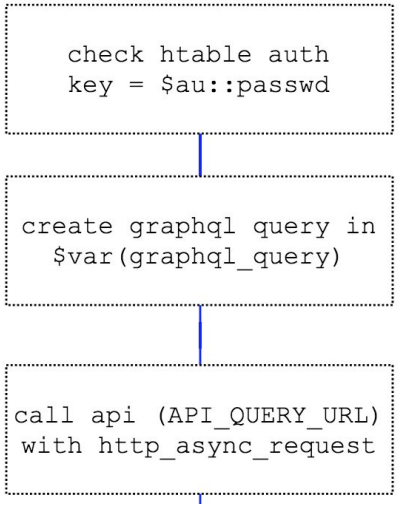
- we call API to retrieve user profile and store it in htable
- we use `pv_auth_check` method to deal with authentication
- Orchestrator calls via RPC kamailio if profile changes and need to be reloaded / updated

Caching authentication in this way improves performance and optimize API calls

Kamailio ASAP - Auth module

How we implemented authentication in kamailio ?

```
route[AUTHCACHE] {
      if($au && $sht(auth=>$au::passwd)==$null) {
      # get password from API
      ...
      $var(graphql_query) = "{\"query\": \"{ kamailio
{ subscriber { auth(username:\\\"" + $au + "\\\") {
username\\n secret } } } } \"}";
      $http_req(body) = $var(graphql_query);
      http_async_query(API_QUERY_URL, "SET_PASSWD");

...

    }
```

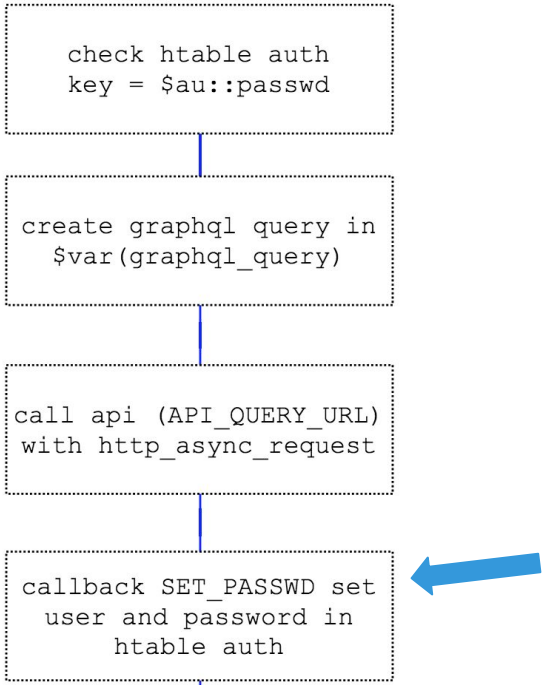


evosip
the world outloud

Kamailio ASAP - Auth module

How we implemented authentication in kamailio ?

```
route[SET_PASSWD] {
  if ($http_ok && $http_rs == 200) {
    xlog("L_INFO", "route[SET_PASSWD]: response
$http_rb)\n");

jansson_get("data.kamailio.subscriber.auth.username",
$http_rb, "$var(username)");

jansson_get("data.kamailio.subscriber.auth.secret",
$http_rb, "$var(secret)");
    xlog("L_INFO", "route[SET_PASSWD]: setting
password >>>$var(secret)<<< for user
>>>$var(username)<<< in shared table\n");
    $sht(auth=>$var(username)::passwd) =
$var(secret);
  } else {
      xlog("L_INFO", "route[HTTP_REPLY]: error
$http_err)\n");
  }
}
```

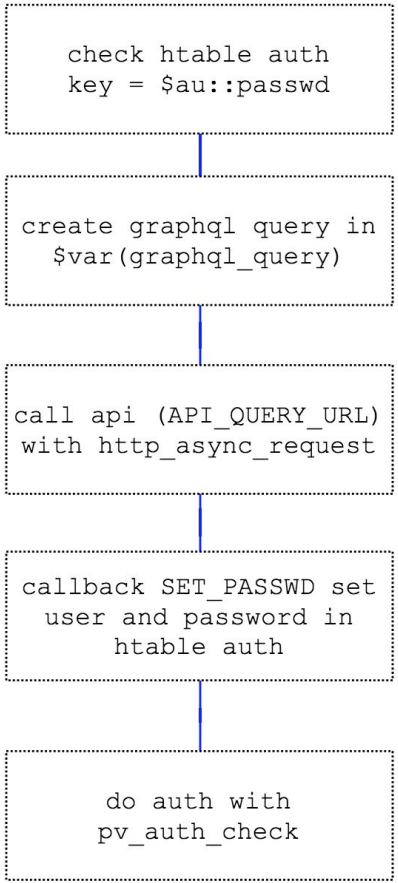


evosip
the world outloud

Kamailio ASAP - Auth module

How we implemented authentication in kamailio ?

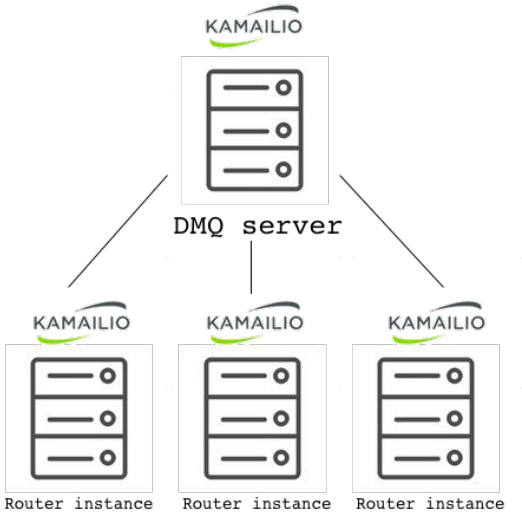
```
 $var(user_passwd) = $sht(auth=>$au::passwd);
    if(!pv_auth_check("$fd", "$var(user_passwd)",
"0", "1")) {
        auth_challenge("$fd", "1");
        exit;
    }#end if
} # end route[AUTH_ACCOUNT]
```



evosip
the world outloud

Kamailio ASAP - Usrloc module

We save in memory and share user locations using **DMQ**
As of the stateless and mutational architecture we decided to use a containerized DMQ server based on kamailio



Router instance

```
…
loadmodule "dmq.so"
loadmodule "usrloc.so"
loadmodule "dmq_usrloc.so"
#!define DMQ_SERVER_ADDRESS "sip:<pod-macvlan-ip>:5060"
#!define DMQ_NOTIFICATION_ADDRESS "sip:dmq-service:5060"
modparam("dmq_usrloc", "enable", 1)
modparam("dmq", "multi_notify", 1)
modparam("dmq", "num_workers", 4)
```

DMQ server

```
...
request_route {
  if(is_method("KDMQ")){
    dmq_handle_message();
  }
...
```

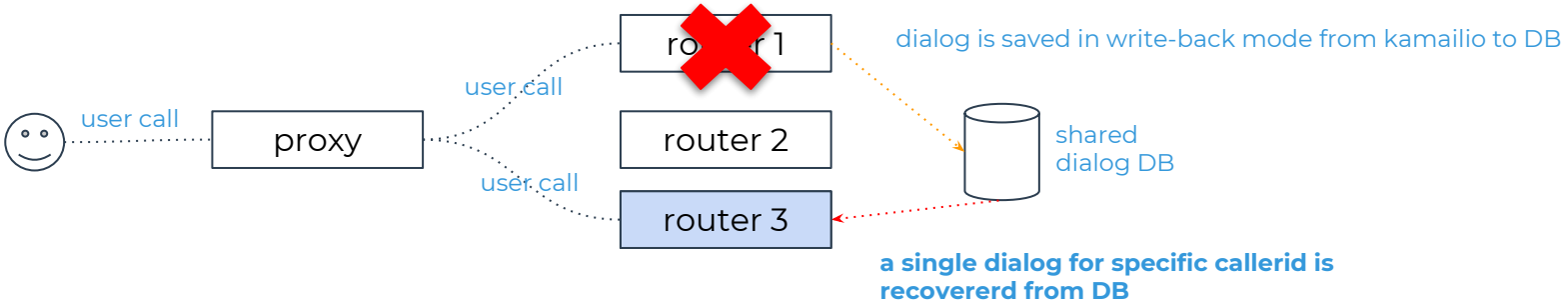

evosip
the world outloud

Kamailio ASAP - Dialogs

We thought that having tons of dialogs replicated among router instances was not the best way to achieve an unlimited scalable architecture

The best way to deal with dialogs in a distributed form for us is:

- every router instance deals with its own dialogs
- foreign dialogs are managed in case of tear-down or failover
- foreign dialogs have to be manager “on demand”



Kamailio ASAP - Dialogs

We needed to implement an intelligent and distributed sharing of dialogs in order to work with a fast architecture mutation and failover-proof

Now kamailio has a new amazing method implemented in dialog module

dlg_db_load_callid("$ci")

(ref. https://github.com/kamailio/kamailio/issues/1512)

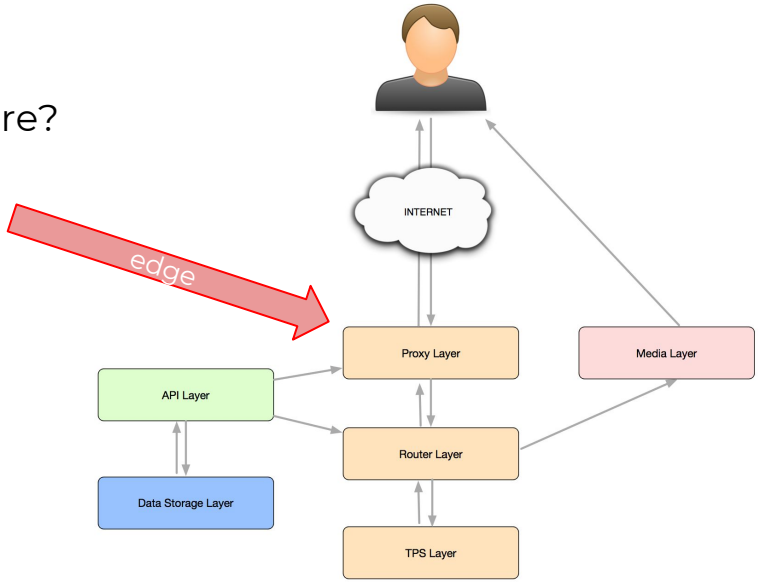
```
...
if(has_totag()) {
    if(!is_known_dlg()) {
     # not a MINE dialog, let's recover from DB
        dlg_db_load_callid("$ci");
          ....
        dlg_manage();
    }
}
...
```

HA and balancing

Possible solutions in a distributed architecture?

DNS

- Round-robin
- GeoIP
- short TTL



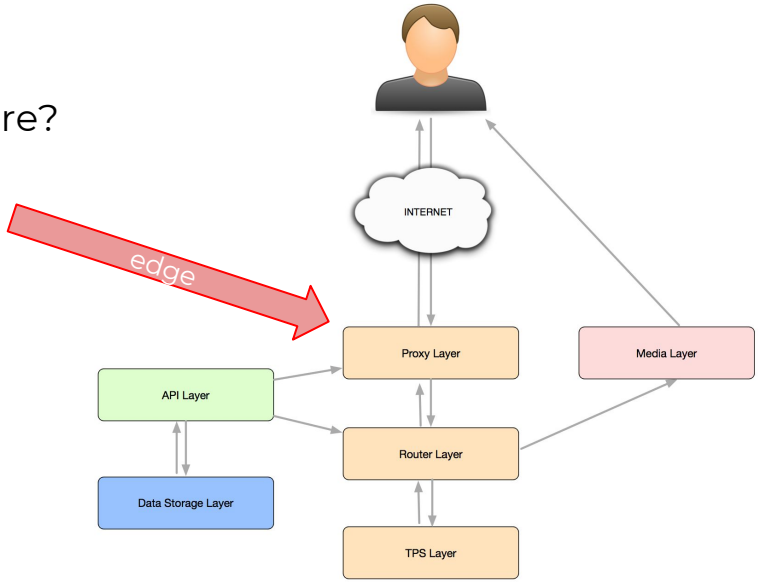
HA and balancing

Possible solutions in a distributed architecture?

DNS

- [redacted]

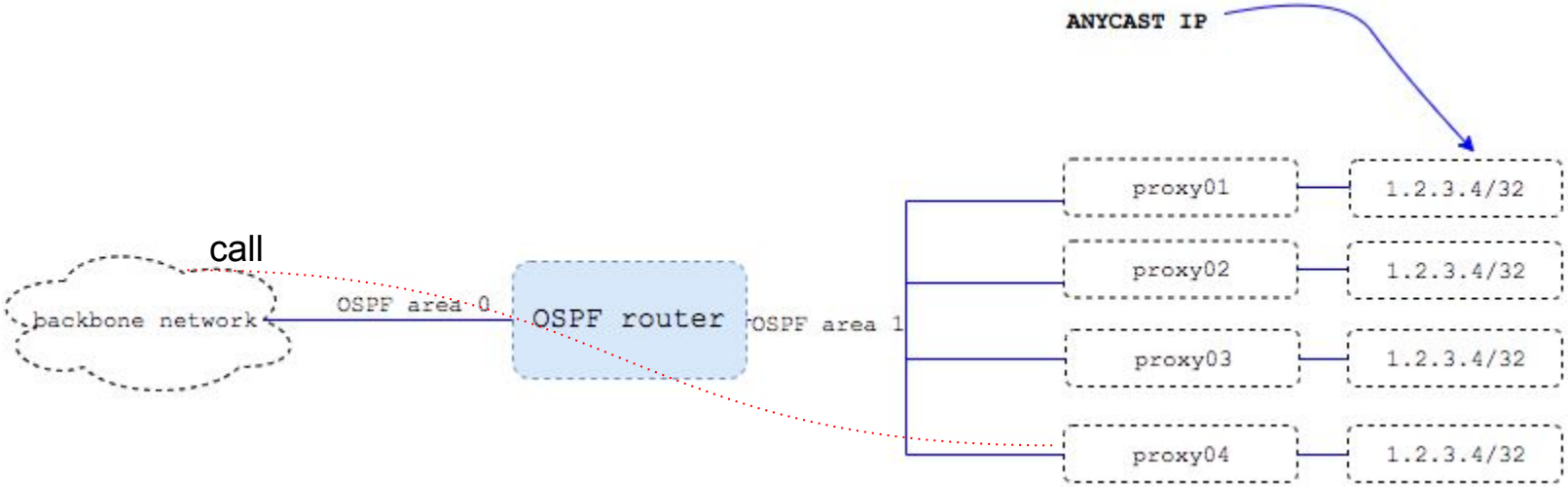
application / service **caching** !



Anycast IP

- using dynamic routing (OSPF)
  - proxy instances
  - border routers

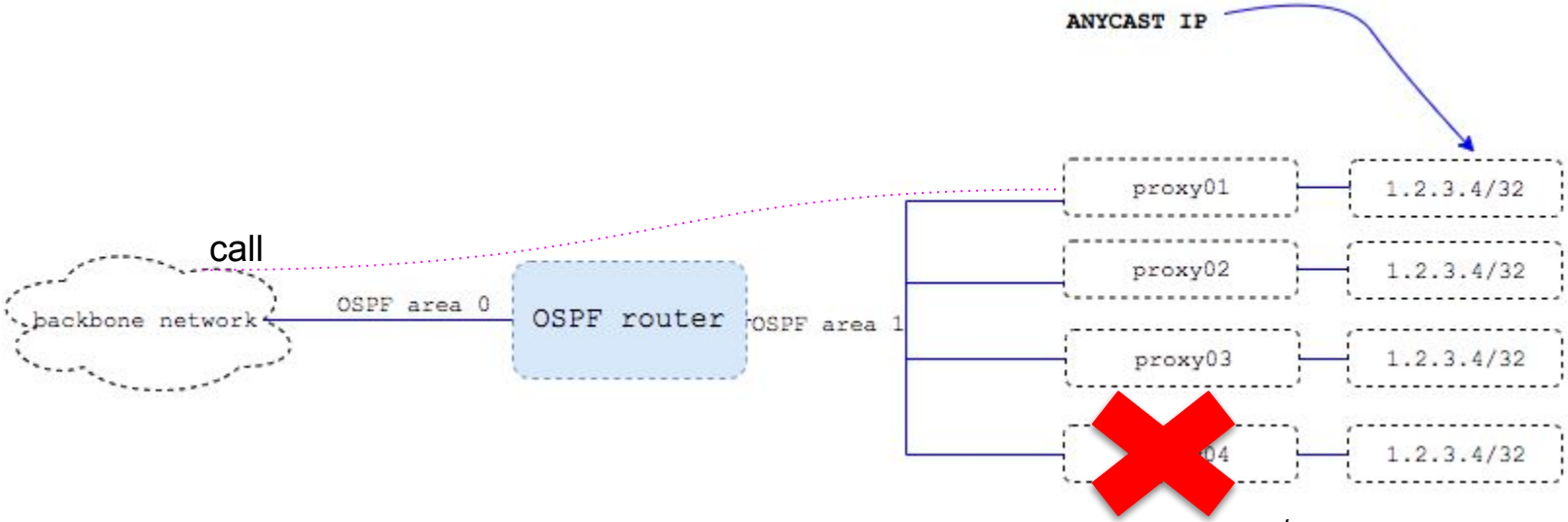
HA and balancing (again in containers)



We use OSPF equal cost multipath and balance hashing per source / destination packets (a session with specific client will maintain the path to the same proxy)

Reducing OSPF timeouts in “area 1” gave us the possibility to converge more quickly in case of tear-down or failover

HA and balancing (again in containers)

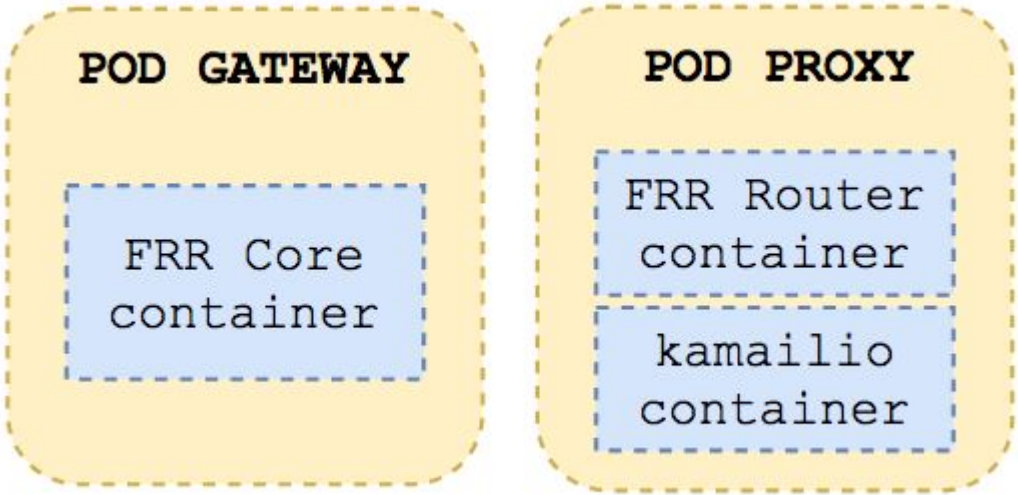


We use OSPF equal cost multipath and balance hashing per source / destination packets (a session with specific client will maintain the path to the same proxy)

Reducing OSPF timeouts in “area 1” gave us the possibility to converge more quickly in case of tear-down or failover

HA and balancing (again in containers)

We decided to use FRRouting (https://frrouting.org/ a fork of the famous Quagga project) both for core ospf router and for proxy instances that announce the anycast ip



HA and balancing (again in containers)

FRR configuration example

```
FRR CORE
...
interface net0
 description *** OSPF MacVLAN ***
 ip ospf dead-interval minimal hello-multiplier 5
router ospf
 default-information originate always
 passive-interface default
 no passive-interface net0
 no passive-interface eth0
 network <MACVLAN_NET>/<MACVLAN_SUB> area 1
 network <BACKBONE_NET>/<BACKBONE_SUB> area 0

 ip route 0.0.0.0/0 <DEFAULT_GW>
```

we set a short dead-interval to reach a fast convergence in “area 1”

this router is the default gateway for proxy layer

we define areas
- eth0 as backbone area 0
- net0 as proxy area 1

HA and balancing (again in containers)

FRR configuration example

```
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>
```

we set a short dead-interval to reach a fast convergence in “area 1”

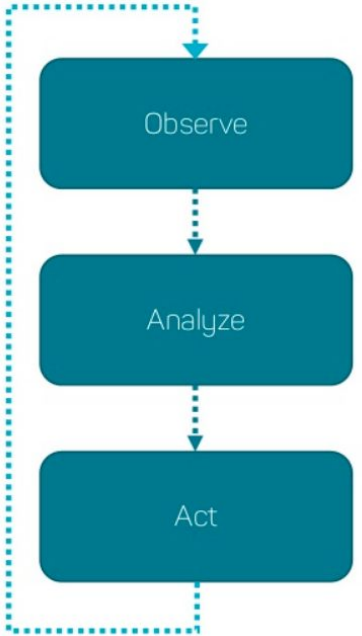
we create the static route to Null0 to annunce the **anycast ip**

Orchestrate !

And what about population of dispatchers and RTP nodes ?

We wrote a kubernetes **controller** to notify events (create, update, delete) among the pods

These notifications are sent to a "sidecar" container which in turn updates the dispatcher lists or dbtexts and call kamailio through RPC to trigger a reload



Orchestrate !

Module reload in kamailio using **xhttp**

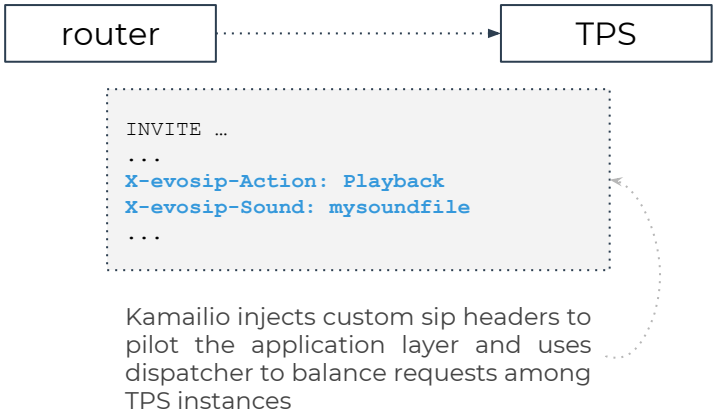
```
event_route[xhttp:request] {
  ...
  if ($hu =~ "^/rpc") {
    xlog("L_NOTICE", "[XHTTP:REQUEST] $si ACCEPTED ***\n");
    jansson_get("method", "$rb", "$var(rpcMethod)");
    xlog("L_NOTICE", "[XHTTP:REQUEST] RPC METHOD: $var(rpcMethod) ***\n");

    if($var(rpcMethod) == "dispatcher.reload") {
      xlog("L_NOTICE", "Reloading dispatcher list\n");
      python_exec("updateDispatchers");
      CHECK_XHTTP_EXIT
    } else if($var(rpcMethod) == "rtpengine.reload") {
      xlog("L_NOTICE", "Reloading RTP list\n");
      python_exec("updateRTPs");
      CHECK_XHTTP_EXIT
    } else if($var(rpcMethod) == "permissions.addressReload") {
      xlog("L_NOTICE", "Reloading address list\n");
    } else if($var(rpcMethod) == "remove.dispatcher") {
      if($hdr(dispatcherIP) != "") {
        xlog("L_NOTICE","RPC Call: remove dispatcher: $hdr(dispatcherIP)\n");
```

Application layer - stateless

We are using Asterisk as a **stateless application**

we use SIP headers to instruct asterisk what is the application with parameters to execute

this makes evosip able to use multiple application services (asterisk, freeswitch, sems, etc) without changing anything on kamailio side



asterisk extensions.conf

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

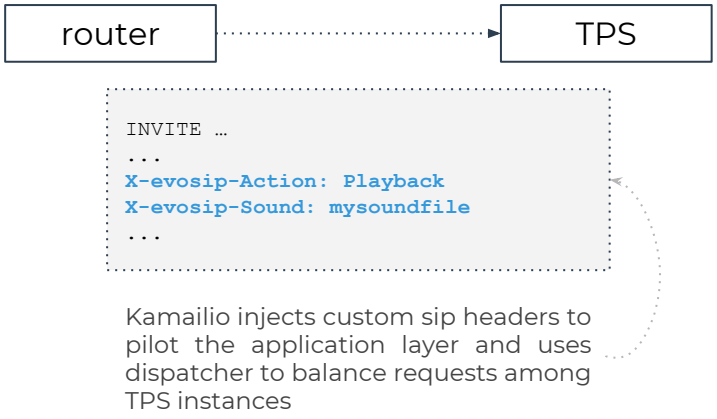
Asterisk uses extensions modules and default context to dispatch requests using GotoIf statement depending on the x-evosip-Action header

evosip
the world outloud

Application layer - stateless

We are using Asterisk as a **stateless application**

playback, transcoding, voicemail and other applications have its own portion of context



asterisk extensions.conf

```
[default]
…
exten => _X.,n(playback),NoOp(*** Doing playback - ***)
;PATH of sound file
;${SIP_HEADER(X-evosip-SoundCustom)}${SIP_HEADER(X-evosip-SoundLocale)}
${SIP_HEADER(X-evosip-Sound)}
exten => _X.,n,SipRemoveHeader(X-evosip)
exten => _X.,n,Progress()
exten => _X.,n,Wait(1)
exten =>
_X.,n,Set(exists=${STAT(e,${ASTDATADIR}/sounds/${SIP_HEADER(X-evosip-So
undCustom)}/${SIP_HEADER(X-evosip-SoundLocale)}/${SIP_HEADER(X-evosip-S
ound)}.wav)})
exten => _X.,n,Playback(${IF($[ ${exists} = 1 ] ?
${SIP_HEADER(X-evosip-SoundCustom)}/${SIP_HEADER(X-evosip-SoundLocale)}
/${SIP_HEADER(X-evosip-Sound)},noanswer :
default/${SIP_HEADER(X-evosip-SoundLocale)}/${SIP_HEADER(X-evosip-Sound
)},noanswer)})
; exten => _X.,n,Playback(${SIP_HEADER(X-evosip-Option)},noanswer)
exten => _X.,n,Hangup()
```

Application layer - stateless

there is no sip trunk or peer, everything is **profile-less** (and uses the default context)

```
Connected to Asterisk 13.1.0~dfsg-1.1ubuntu4.1 currently running on tps-d768ccb6b-bnnlk (pid = 73)
tps-d768ccb6b-bnnlk*CLI> sip show peers
Name/username             Host                                    Dyn Forcerport Comedia    ACL Port
Status      Description
0 sip peers [Monitored: 0 online, 0 offline Unmonitored: 0 online, 0 offline]
```

SDP manipulations are made using context command “Set(SIP_CODEC)“

Media can be in cloud or on premise

We use RTPEngine to bridge RTP traffic

To enhance the QoE of voice / video services you are able to move closer to you this layer; less latency, less network hops and full control of your media in your network!

In **evosip** rtpengine works in kubernetes using kernel module **xt_RTPENGINE** and scaling automatically new instances (also on the same host)

Every node (that shares the same kernel in every container) loads at startup the **xt_RTPENGINE** module and every instance, in bootstrap mode, uses the first free "**table**" on that node (and uses IPTABLES inside the container to mark packets)

Media can be in cloud or on premise

example of rtp instance bootstrap bash script:

```
# configure iptables fw
iptables -N rtpengine 2> /dev/null
iptables -D INPUT -j rtpengine 2> /dev/null
iptables -I INPUT -j rtpengine
iptables -D rtpengine -p udp -j RTPENGINE --id " $TABLE" 2>/dev/null
iptables -I rtpengine -p udp -j RTPENGINE --id " $TABLE"
```

Media can be in cloud or on premise

example of rtp instance bootstrap bash script:

```
cat << EOF > /etc/rtpengine/rtpengine.conf

[rtpengine]

table = $TABLE

interface = $POD_PUBLIC_IP

#interface=internal/$POD_PUBLIC_IP;external/$POD_PUBLIC_IP

... snip ...

EOF

# run rtpengine

/usr/sbin/rtpengine --table=$TABLE --config-file=/etc/rtpengine/rtpengine.conf
--pidfile=/var/run/rtpengine.pid -E -f -F

/pod_scripts/disablePod.sh 99 "rtpengine crashed"
```

Recap

evosip is a startup project

ideas, contributions, partnership and knowledge sharing are really welcome !

What we focused on:

- scalability (auto)
- distribution
- QoE
- being ASAP (as stateless as possible)

Recap

evosip is a startup project

ideas, contributions, partnership and knowledge sharing are really welcome !

What will be in the next months:

- datacollect with ELK stack and Homer
- preemptive autoscaling with machine learning
- chaos engineering
- IPV6 protocol

Stay tuned !

evosip
the world outloud

Do you like concepts and examples in this speech ?

subscribe and join evosip community @ http://evosip.cloud

Articles, news, interviews, podcast and videos of the project for free implementing the knowledge sharing!