

Kamailio loadbalancerként működne több asterisk szerver előtt. Hogyan lehet ezt megvalósítani

Ez egy klasszikus és erős architektúra: **Kamailio mint SIP load balancer** több **Asterisk szerver** előtt.

Az elv lényege, hogy a Kamailio a beérkező SIP forgalmat (INVITE, REGISTER stb.) több Asterisk szerver között osztja el — dinamikusan vagy statikusan, adott szabályok alapján.

Nézzük lépésről lépésre, hogyan lehet ezt megvalósítani:

## 1. Alapfelállítás

Komponensek:

- **Kamailio** → SIP proxy és load balancer
- **Asterisk1, Asterisk2, ... AsteriskN** → media és signalling szerverek
- (opcionálisan: adatbázis pl. MySQL/PostgreSQL, ha Kamailio load balancer modult használsz)

## 2. Szükséges Kamailio modulok

A következő modulokra lesz szükség:

```
bash
```

[Kód másolása](#)

```
loadmodule "dispatcher.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
```

A `dispatcher` modul a legfontosabb — ez kezeli a load balancingot.

## 3. Dispatcher konfigurálása

`dispatcherlist` fájl (pl. `/etc/kamailio/dispatcher.list`):

```
bash
```

[Kód másolása](#)


```
1 sip:10.0.0.11:5060 0 weight=50 description="Asterisk1"
1 sip:10.0.0.12:5060 0 weight=50 description="Asterisk2"
```

Itt a 1 a csoportazonosító — több különböző célcsoport is lehet (pl. regisztrációs szerverek, media szerverek stb.).

## 4. Kamailio.cfg – logika hozzáadása

A request\_route szekcióban (ahol az INVITE vagy REGISTER üzeneteket kezeled), illeszd be:

kamailio

 Kód másolása

```
# Load balancing for INVITE requests
if (is_method("INVITE")) {
    if (!ds_select_dst("1", "4")) {
        sl_send_reply("500", "No Destination");
        exit;
    }
    xlog("L_INFO", "Dispatching call to <$du>\n");
    route(RELAY);
    exit;
}
```

Itt a ds\_select\_dst("1", "4") azt jelenti:

- „1” → dispatcher group 1
- „4” → load balancing módszer (round-robin + weight)

További módszerek is vannak (hash, callid, stb.), hogy stabilabb legyen a routing (pl. adott hívó mindig ugyanarra az Asteriskre menjen).


## 5. REGISTER kérések kezelése

Ha a Kamailio-n keresztül regisztrálnak a kliensek, akkor:

- Kamailio fogadja a REGISTER-t,
- Eltárolhatja a location adatbázisban,
- Majd továbbíthatja az egyik Asterisk felé.

Alap routing logika:

kamailio

 Kód másolása

```
if (is_method("REGISTER")) {
    save("location");
    # opcionális: load balancing asterisk felé
    if (!ds_select_dst("1", "4")) {
        sl_send_reply("500", "No Destination");
        exit;
    }
    route(RELAY);
    exit;
}
```


**Tipp:** Ha Asteriskek nem osztanak közös `astdb`-t vagy realtime DB-t, a regisztrációk szinkronizálása bonyolódhat.  
Jobb, ha Kamailio kezeli a regisztrációkat, és az Asteriskek csak *back-end call terminálok*.

---

## 6. Dispatcher dinamikus menedzsment

A dispatcher lista frissíthető runtime-ban is:

```
bash
```

 Kód másolása

```
kamcmd dispatcher.reload  
kamcmd dispatcher.list
```

Ezen keresztül új Asterisk node is hozzáadható, vagy egy kiesett node deaktiválható.


---

## 7. Health check és failover

A `dispatcher` modul képes **pingelni** (OPTIONS üzenettel) az Asteriskeket, hogy azok elérhetők-e:

`kamailio.cfg` -ben:

```
kamailio
```

 Kód másolása


```
modparam("dispatcher", "ds_ping_method", "OPTIONS")  
modparam("dispatcher", "ds_ping_interval", 30)  
modparam("dispatcher", "ds_probing_mode", 1)  
modparam("dispatcher", "ds_ping_from", "sip:kamailio@yourdomain.com")
```

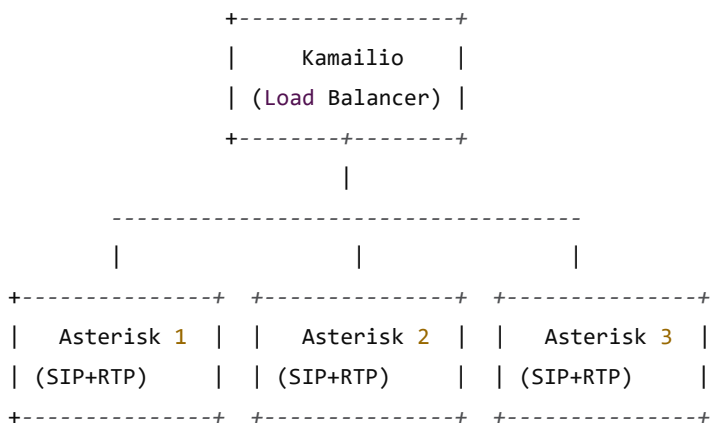
Ha egy szerver nem válaszol, automatikusan inaktívvá válik, és a forgalom a többi szerverre terelődik.

---

## 8. Példa topológia

```
pgsql
```

 Kód másolása



## 🌱 9. Opcionális továbbfejlesztések

- RTPEngine / RTPProxy integrálása, ha a médiaforgalmat is át akarod vezetni.
- Dispatcher + DMQ: több Kamailio node load balancing clusterhez.
- SIP over TLS / WSS: ha WebRTC-t is szeretnél bevonni.
- Asterisk Realtime: közös MySQL backend a több Asterisk között (ha regisztrációkat is kezelni akarsz backend oldalon).

Ha leírod, hogy **milyen SIP forgalmat szeretnél load balance-olni** (pl. csak hívások, vagy regisztrációk is), tudok adni egy **konkrét Kamailio.cfg mintát**, amit kiindulásként használhatsz.

Szeretném, hogy adjak egy működő példakonfigurációt is (dispatcher + alap INVITE routinggal)?

📄 👍 💬 ↗️ ↻️ ...

Eddig az asterisk szerveren volt minden, a regisztráció és a hívás logika is. De több asterisk lesz a kliensek számának növekedése miatt. Ezért a több asterisk szerver elé kerül a kamailio. A kamailio végezné a regisztrációt, a hívás logika továbbra is maradna az asterisk szervereken. Az asterisk szerver is adatbázishoz kapcsolódik.

Tökéletes — ez az egyik **legjobb és leggyakoribb SIP-architektúra**, amikor a rendszer skálázása a cél.

Amit leírsz, az a klasszikus "**Kamailio front-end + Asterisk back-end**" modell, ahol:

- Kamailio kezeli az összes **regisztrációt, NAT-ot, load balancingot, routingot**.
- Asterisk csak a **híváslogikát és media feldolgozást** végzi (dialplan, IVR, queue, stb.).
- Mind az Asterisk, mind Kamailio ugyanarra az **adatbázisra** támaszkodik (pl. MySQL), de eltérő táblákon.

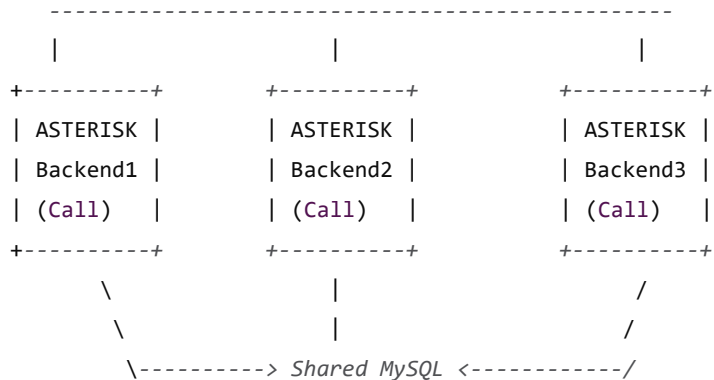
Lépünk végig a megvalósításon:

## ⚙️ ARCHITEKTÚRA ÁTTEKINTÉS

pgsql

📄 Kód másolása

```
+-----+
|          KAMAILIO          |
| - REGISTER kezelése      |
| - Load balancing         |
| - Failover / NAT fix      |
+-----+
|
```




## 1. Kamailio kezeli a REGISZTRÁCIÓT

A kliensek a Kamailio-hoz csatlakoznak.

Kamailio elmenti a regisztrációkat **usrloc/location táblába**, ami egy adatbázisban van.

### Példa modul betöltés:


bash

 Kód másolása

```
loadmodule "db_mysql.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "dispatcher.so"
```

### Modulparaméterek:


bash

 Kód másolása

```
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "db_url", "mysql://kamailio:pass@localhost/kamailio")
modparam("registrar", "method_filtering", 1)
```

## 2. REGISTER logika

kamailio

 Kód másolása

```
if (is_method("REGISTER")) {
    if (!www_authorize("domain.hu", "subscriber")) {
        www_challenge("domain.hu", "0");
        exit;
    }
    if (!save("location")) {
        sl_reply_error();
        exit;
    }
    sl_send_reply("200", "OK");
}
```

```
    exit;
}
```

👉 Ezzel a Kamailio adatbázisban tárolja a regisztrációkat, és az Asteriskek már nem a saját regisztrációs adatbázisukat használják, hanem Kamailio backendjét olvassák (Realtime userlocation query).

---

### 3. Asterisk oldali beállítás

Az Asteriskeknek meg kell tudniuk, hova van regisztrálva egy kliens.

Erre több opció van:


#### ◆ a) Asterisk Realtime integration

A Kamailio által használt location tábla olvasható az Asteriskből:

extconfig.conf

```
ini


sipusers => odb,asterisk,sipusers
sippeers => odb,asterisk,sippeers
```

 Kód másolása

sorcery.conf (ha PJSIP)

```
ini

[res_pjsip]
endpoint=realtime,ps_endpoints
aor=realtime,ps_aors
auth=realtime,ps_auths
```

 Kód másolása

Igy az Asterisk mindig a közös adatbázisból olvassa, melyik kliens milyen IP-re van regisztrálva.


---

### 4. Kamailio load balancing INVITE esetén

A Kamailio a beérkező INVITE üzeneteket load balance-olja az Asteriskek felé.

```
kamailio


if (is_method("INVITE")) {
    # load balancing Asterisk cluster között
    if (!ds_select_dst("1", "4")) {
        sl_send_reply("500", "No available backend");
        exit;
    }
    xlog("L_INFO", "Routing call to $du\n");
    route(RELAY);
}
```

 Kód másolása

```
    exit;  
}
```

dispatcher.list:

arduino

 Kód másolása


```
1 sip:10.0.0.11:5060 0 weight=50 description="Asterisk1"  
1 sip:10.0.0.12:5060 0 weight=50 description="Asterisk2"  
1 sip:10.0.0.13:5060 0 weight=50 description="Asterisk3"
```

## 5. Kamailio ↔ Asterisk kommunikáció

Az Asteriskeket úgy állítod be, hogy Kamailio-tól fogadjanak el hívásokat, ne közvetlenül a kliensektől.

pjsip.conf példa:

ini


 Kód másolása

```
[trunk-kamailio]  
type=endpoint  
transport=transport-udp  
context=from-kamailio  
disallow=all  
allow=alaw,ulaw  
aors=kamailio  
  
[kamailio]  
type=aor  
contact=sip:10.0.0.10:5060 ; Kamailio IP
```

## 6. Dispatcher health-check

Kamailio automatikusan monitorozhatja az Asteriskeket:

kamailio

 Kód másolása

```
modparam("dispatcher", "ds_ping_method", "OPTIONS")  
modparam("dispatcher", "ds_ping_interval", 30)  
modparam("dispatcher", "ds_probing_mode", 1)
```

Így ha egy Asterisk leáll, Kamailio nem küld rá hívásokat.

## 7. Adatbázis megosztás

- Kamailio és Asterisk közös **MySQL adatbázisban** oszthat meg adatokat.
- Kamailio: subscriber , location táblák

- Asterisk: ps\_endpoints , ps\_aors , ps\_auths , cdr , stb.
- Ezek elválaszthatók ugyanabban az adatbázisban, vagy külön DB-ben is lehetnek.

## ✅ Összefoglalás

Funkció	Komponens
Regisztrációk kezelése	Kamailio
Load balancing	Kamailio (dispatcher)
Híváslogika (dialplan, IVR, queue)	Asterisk backend
Adatbázis / Realtime userinfo	Megosztott MySQL
Health-check, failover	Kamailio dispatcher

Ha szeretnéd, tudok adni egy **komplett konfiguráció-csomagot** (Kamailio.cfg + dispatcher.list + Asterisk PJSIP példa + adatbázis sémák) ami **működőképes load balancer + regisztrációs proxy** architektúrát hoz létre.

👉 Szeretnéd, hogy generáljam ezt a példát (Kamailio + 2 Asterisk back-enddel)?

📄 👍 💬 ⬆️ ↺ ...

igen, generáld

Nagyon jó — akkor készítek neked egy **működő példát**:

- 👉 Kamailio mint SIP proxy és load balancer,
- 👉 2 Asterisk backend szerverrel,
- 👉 megosztott MySQL adatbázissal (Kamailio kezeli a regisztrációkat, Asterisk olvassa őket Realtime-ban).

A cél: minden kliens **Kamailio-hoz** regisztrál, de a hívások **Asteriskeken** zajlanak. A Kamailio automatikusan **elosztja a hívásokat** a backendek között.

## 🧱 RENDSZER ÁTTEKINTÉS


Komponens	IP	Szerep
Kamailio	10.0.0.10	SIP proxy, registrar, load balancer
Asterisk 1	10.0.0.11	backend híváslogika
Asterisk 2	10.0.0.12	backend híváslogika



Komponens	IP	Szerep
MySQL	10.0.0.13	közös adatbázis (user, location, realtime)

## 1. kamailio.cfg – fő konfiguráció

kamailio

 Kód másolása

```
##### Kamailio fő konfiguráció #####

#define DBURL "mysql://kamailio:password@10.0.0.13/kamailio"

##### Modulok betöltése #####
loadmodule "db_mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "registrar.so"
loadmodule "usrloc.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "dispatcher.so"
loadmodule "textops.so"
loadmodule "xlog.so"

##### Modulparaméterek #####
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "db_url", DBURL)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")

modparam("dispatcher", "list_file", "/etc/kamailio/dispatcher.list")
modparam("dispatcher", "ds_ping_method", "OPTIONS")
modparam("dispatcher", "ds_ping_interval", 30)
modparam("dispatcher", "ds_probing_mode", 1)
modparam("dispatcher", "ds_ping_from", "sip:kamailio@domain.hu")

##### Alap routing #####
request_route {
    xlog("L_INFO", "--- New request: $rm from $si ---\n");

    # REGISTER
    if (is_method("REGISTER")) {
        if (!www_authorize("domain.hu", "subscriber")) {
            www_challenge("domain.hu", "0");
            exit;
        }
        if (!save("location")) {
            sl_send_reply("500", "Registration failed");
        }
    }
}
```

```

        exit;
    }
    sl_send_reply("200", "OK");
    exit;
}

# INVITE - load balancing
if (is_method("INVITE")) {
    if (!lookup("location")) {
        if (!ds_select_dst("1", "4")) {
            sl_send_reply("500", "No destination available");
            exit;
        }
        xlog("L_INFO", "Routing call to backend: $du\n");
        route(RELAY);
        exit;
    } else {
        route(RELAY);
        exit;
    }
}

# Egyéb SIP üzenetek
if (is_method("OPTIONS")) {
    sl_send_reply("200", "Keepalive OK");
    exit;
}

route(RELAY);
}


route[RELAY] {
    record_route();
    if (!t_relay()) {
        sl_reply_error();
    }
    exit;
}

```

---

## 2. dispatcher.list

bash

 Kód másolása

```

1 sip:10.0.0.11:5060 0 weight=50 description="Asterisk1"
1 sip:10.0.0.12:5060 0 weight=50 description="Asterisk2"

```

---


## 3. MySQL adatbázis

Kamailio adatbázis sémák (pl. kamdbctl create létrehozza):

- subscriber – felhasználók
- location – regisztrációs helyek
- version – verziók

A subscriber táblába például így vehetsz fel user-t:

sql


 Kód másolása

```
INSERT INTO subscriber (username, domain, password) VALUES ('1001', 'domain.hu',  
INSERT INTO subscriber (username, domain, password) VALUES ('1002', 'domain.hu',
```

## 4. Asterisk beállítások

pjsip.conf


ini

 Kód másolása

```
[transport-udp]  
type=transport  
protocol=udp  
bind=0.0.0.0:5060  
  
; kapcsolat Kamailio felé  
[kamilio-trunk]  
type=endpoint  
transport=transport-udp  
context=from-kamilio  
disallow=all  
allow=alaw,ulaw  
aors=kamilio-trunk  
  
[kamilio-trunk]  
type=aor  
contact=sip:10.0.0.10:5060  
  
; Realtime user betöltés  
#include pjsip_realtime.conf
```

extconfig.conf

ini

 Kód másolása

```
ps_endpoints => odbc,asterisk,ps_endpoints  
ps_aors => odbc,asterisk,ps_aors  
ps_auths => odbc,asterisk,ps_auths
```

## 5. Asterisk dialplan (részlet extensions.conf)

ini


```
[from-kamailio]
exten => _X.,1,NoOp(Incoming call from Kamailio)
same => n,Dial(PJSIP/${EXTEN},20)
same => n,Hangup()
```

---

## 6. ODBC / MySQL kapcsolat (Asterisk)

### res\_odbc.conf


ini

 Kód másolása

```
[asterisk]
enabled => yes
dsn => MySQL-asterisk
username => asterisk
password => password
pre-connect => yes
```

### odbc.ini

ini

 Kód másolása


```
[MySQL-asterisk]
Description=MySQL connection
Driver=MySQL
Server=10.0.0.13
Database=kamailio
User=asterisk
Password=password
Port=3306
```

---

## 7. Kapcsolat tesztelése

### 1. Kamailio-ban ellenőrzés:

bash

 Kód másolása

```
kamcmd dispatcher.list
kamcmd ul.dump
```

2. Regisztráció után látni fogod a klienst a `location` táblában.

3. Hívás indításkor Kamailio a hívást elosztja az egyik Asteriskre (round robin).

---

## Összefoglaló működés

1. A kliensek **Kamailio-hoz** regisztrálnak.
2. Kamailio eltárolja az adatokat MySQL-ben.

3. Az Asteriszek ugyanebből a MySQL adatbázisból olvassák a `ps_aors` , `ps_endpoints` stb. adatokat.
  4. Kamailio load-balance-olja az INVITE kéréseket az aktív Asteriszek között.
  5. Ha egy Asterisk nem válaszol (OPTIONS probe), automatikusan kiveszi a listából.
- 

Szeretném, hogy készítsen hozzá egy **teljesen használható Kamailio + MySQL docker-compose példát** is (3 konténer: kamailio + mysql + 2 asterisk)?