Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Confronto tra Ucarp, Keepalived, Corosync e Pacemaker per la gestione di servizi

Antonio Francesco Gentile, Davide Macrì, Emilio Greco

RT-ICAR-CS-25-01 **Gennaio 2025**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: www.icar.cnr.it
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.icar.cnr.it
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: www.icar.cnr.it

Indice generale

Introduzione

Questo rapporto tecnico presenta un'analisi comparativa tra quattro strumenti principali per la gestione dell'alta disponibilità (HA) in ambienti Linux: **Ucarp**, **Keepalived**, **Corosync**, e **Pacemaker**. L'obiettivo del documento è fornire una guida dettagliata per comprendere le caratteristiche, le funzionalità e i casi d'uso di ciascuna soluzione, supportando la scelta della tecnologia più adeguata alle diverse esigenze operative.

Gli strumenti analizzati sono utilizzati in scenari critici dove la continuità del servizio è fondamentale, come nella gestione di cluster, bilanciamento del carico e failover di risorse di rete o applicative. La relazione approfondisce i seguenti aspetti:

- **Caratteristiche principali**: Analisi delle funzionalità di base e avanzate, come il supporto per il failover, il bilanciamento del carico, la gestione delle risorse e la comunicazione tra nodi.
- **Casi d'uso applicativi**: Implementazioni pratiche di ciascuna soluzione in contesti reali, tra cui la gestione di server Nginx, database MariaDB, sistemi di file NFS e ambienti IoT basati su OpenWrt.
- **Confronto tecnico**: Valutazione comparativa delle capacità di scalabilità, semplicità di configurazione, integrazione e flessibilità, per aiutare i lettori a identificare il tool più adatto a specifici requisiti.

Un'attenzione particolare è riservata all'integrazione tra **Corosync** e **Pacemaker**, che insieme costituiscono una soluzione robusta per scenari complessi e ad alta criticità. Inoltre, il rapporto include un esempio concreto di utilizzo in ambienti IoT, evidenziando come configurare router OpenWrt in modalità alta disponibilità.

Ucarp

Ucarp (**User Controlled Automatic Redundancy Protocol**) è un'implementazione userland del protocollo CARP[1] (Common Address Redundancy Protocol), progettata per fornire alta disponibilità nelle configurazioni di rete. Il suo scopo principale è gestire il failover delle interfacce di rete, garantendo che, in caso di guasto di un nodo, un altro possa subentrare senza interruzioni significative del servizio.

Una delle caratteristiche distintive di Ucarp è la sua semplicità. La configurazione e la gestione risultano intuitive, rendendolo particolarmente adatto per ambienti meno complessi o per amministratori di sistema che preferiscono soluzioni leggere. Il protocollo supporta sia IPv4 che IPv6, assicurando compatibilità con diverse infrastrutture di rete. Tuttavia, è importante notare che Ucarp è focalizzato principalmente sulla gestione delle interfacce di rete e non offre funzionalità avanzate per la gestione di applicazioni o servizi a livello di cluster.

I casi d'uso tipici di Ucarp includono scenari in cui è necessaria una gestione semplice ed efficace delle interfacce di rete, come in configurazioni con router e gateway. La sua implementazione può migliorare la resilienza della rete, garantendo che l'indirizzo IP virtuale rimanga disponibile anche in caso di guasto di un nodo.

È fondamentale considerare che Ucarp non dispone di un'integrazione nativa con sistemi di gestione di cluster più complessi, come Pacemaker o Corosync. Pertanto, in ambienti che richiedono una

[1] CARP è un protocollo di rete che consente a più host sulla stessa rete locale di condividere un insieme di indirizzi IP, fornendo ridondanza e failover.

gestione dettagliata delle risorse e delle dipendenze tra servizi, potrebbe essere opportuno valutare soluzioni alternative più robuste.

Di seguito una sintesi delle caratteristiche del protocollo:

- Semplicità: Ucarp è semplice da configurare e gestire, rendendolo adatto per ambienti meno complessi.
- Failover: Fornisce un failover automatico delle interfacce di rete, con un meccanismo di heartbeat per monitorare i nodi.
- Supporto per IPv4 e IPv6: Ucarp supporta entrambi i protocolli IP.
- Limitato a configurazioni di rete: È progettato principalmente per gestire l'alta disponibilità delle interfacce di rete piuttosto che delle applicazioni.
- Casi d'uso: Ideale per piccole reti dove è necessaria una gestione semplice delle interfacce di rete. Può essere utilizzato in scenari con router e gateway.
- Non ha un'integrazione nativa con sistemi di gestione di cluster.
- Può non essere sufficiente per scenari complessi in cui è richiesta una gestione delle risorse più dettagliata.

Keepalived

Keepalived è un software open source progettato per fornire alta disponibilità e bilanciamento del carico in ambienti Linux. Basato sul protocollo VRRP (Virtual Router Redundancy Protocol), Keepalived gestisce indirizzi IP virtuali per garantire la continuità del servizio in caso di guasti hardware o software.

Una delle funzionalità chiave di Keepalived è il supporto per VRRP, che consente a più router o server di condividere un indirizzo IP virtuale. In caso di malfunzionamento del nodo principale, un nodo secondario assume automaticamente il controllo dell'IP virtuale, assicurando la disponibilità continua del servizio.

Oltre al supporto VRRP, Keepalived include meccanismi di health checking per monitorare lo stato dei servizi associati. Questa funzionalità permette di rilevare rapidamente eventuali anomalie nei servizi e di eseguire azioni correttive, come la rimozione di un server non funzionante dal pool di bilanciamento del carico.

Keepalived è spesso utilizzato in combinazione con HAProxy, un popolare software di bilanciamento del carico per applicazioni TCP e HTTP. Questa integrazione consente di creare un'infrastruttura altamente disponibile e scalabile, in cui Keepalived gestisce l'alta disponibilità degli indirizzi IP virtuali e HAProxy distribuisce il traffico tra i server backend.

La configurazione di Keepalived è relativamente semplice e flessibile, permettendo agli amministratori di sistema di adattarlo facilmente alle esigenze specifiche dell'ambiente di rete. Tuttavia, è importante notare che Keepalived è principalmente focalizzato sulla gestione della rete e del bilanciamento del carico e non offre funzionalità avanzate per la gestione delle risorse di cluster, come quelle fornite da strumenti come Pacemaker.

In sintesi, Keepalived è particolarmente utile in ambienti web dove è necessario bilanciare il carico e garantire l'alta disponibilità dei server. La sua integrazione con HAProxy e il supporto per VRRP lo rendono una scelta efficace per la gestione di infrastrutture di rete critiche.

Di seguito una sintesi delle caratteristiche del prodotto:

- Supporto VRRP: Utilizza VRRP per gestire gli indirizzi IP virtuali.
- Health Checking: Include funzionalità di health check per monitorare la salute dei servizi.
- Integrazione con HAProxy: Spesso utilizzato in combinazione con HAProxy per bilanciare il carico delle applicazioni.
- Configurazione semplice: Permette una configurazione relativamente semplice e flessibile.
- Casi d'uso: Particolarmente utile in ambienti web dove è necessario bilanciare il carico e garantire l'alta disponibilità dei server.
- Limitato principalmente a soluzioni di rete e bilanciamento del carico.
- Non offre gestione avanzata delle risorse come Pacemaker.

Corosync

Corosync è un motore di clustering open source che fornisce servizi di comunicazione e gestione per sistemi ad alta disponibilità. Sviluppato per garantire la comunicazione affidabile tra nodi in un cluster, Corosync funge da base per la gestione delle risorse e del quorum, supportando vari moduli, tra cui Pacemaker.

Una delle principali funzionalità di Corosync è la gestione della comunicazione tra nodi attraverso un protocollo ad alte prestazioni, garantendo che i messaggi siano consegnati in modo affidabile e in ordine. Inoltre, Corosync gestisce la membership dei nodi e il quorum, assicurando che solo i nodi autorizzati partecipino al cluster e che le decisioni vengano prese solo quando è presente un numero sufficiente di nodi attivi. Questa gestione è fondamentale per mantenere l'integrità e la coerenza del cluster.

Corosync offre flessibilità supportando diversi sistemi di gestione delle risorse e può essere utilizzato in combinazione con Pacemaker per fornire una soluzione completa di gestione del cluster. Questa combinazione consente di monitorare e gestire le risorse del cluster in modo efficiente, garantendo l'alta disponibilità dei servizi.

I casi d'uso tipici di Corosync includono cluster di server e sistemi di storage distribuiti, dove è essenziale una comunicazione affidabile tra nodi e il monitoraggio dello stato del sistema. La sua capacità di gestire la comunicazione e la membership dei nodi lo rende ideale per ambienti che richiedono alta disponibilità e tolleranza ai guasti.

È importante notare che la configurazione di Corosync può essere più complessa rispetto a soluzioni come Ucarp e Keepalived, richiedendo una comprensione approfondita dei concetti di clustering e delle specifiche esigenze dell'ambiente in cui viene implementato. Tuttavia, questa complessità è giustificata dalla robustezza e dalle funzionalità avanzate che Corosync offre, rendendolo una scelta comune in scenari enterprise dove l'affidabilità e la scalabilità sono fondamentali.

Di seguito una sintesi delle caratteristiche del prodotto:

- Comunicazione tra nodi: Fornisce un protocollo di comunicazione ad alte prestazioni per i nodi del cluster.
- Membership e quorum: Gestisce la membership dei nodi e il quorum per garantire che il cluster funzioni correttamente.
- Flessibilità: Supporta diversi sistemi di gestione delle risorse e può essere utilizzato in combinazione con Pacemaker.

- Casi d'uso: Ideale per cluster di server e sistemi di storage distribuiti, dove è richiesta la comunicazione tra nodi e il monitoraggio della salute.
- Richiede una configurazione più complessa rispetto a Ucarp e Keepalived.
- Spesso utilizzato in scenari enterprise.

Pacemaker

Pacemaker è un gestore di risorse open source progettato per garantire l'alta disponibilità in ambienti cluster. Funziona in combinazione con Corosync, che fornisce servizi di comunicazione tra i nodi del cluster, per monitorare e gestire le risorse, assicurando il failover automatico e il bilanciamento del carico in caso di guasti.

Una delle principali caratteristiche di Pacemaker è la sua capacità di gestire una vasta gamma di risorse, tra cui servizi, file system e qualsiasi altra risorsa configurabile. Consente di definire politiche dettagliate per la gestione delle risorse e delle loro dipendenze, offrendo un controllo granulare sul comportamento del cluster. Questa flessibilità lo rende particolarmente adatto per scenari complessi in cui è necessario garantire l'alta disponibilità di diverse tipologie di applicazioni e servizi, come database e applicazioni critiche.

Tuttavia, è importante notare che la configurazione di Pacemaker richiede una certa complessità e una comprensione approfondita dei concetti di clustering. Sebbene offra una maggiore flessibilità e controllo rispetto ad altri strumenti, l'implementazione efficace di Pacemaker può essere più impegnativa. Per garantire una comunicazione affidabile tra i nodi, Pacemaker funziona in sinergia con Corosync, che gestisce l'appartenenza al cluster e la comunicazione tra i nodi.

In sintesi, Pacemaker è una soluzione potente per la gestione dell'alta disponibilità in ambienti cluster, ideale per scenari che richiedono il monitoraggio e la gestione di diverse risorse con politiche complesse. La sua integrazione con Corosync garantisce una comunicazione efficiente tra i nodi, assicurando la resilienza e la continuità operativa dei servizi critici.

Le caratteristiche principali della soluzione sono:

- Gestione delle risorse: Gestisce le risorse del cluster e offre failover automatico e bilanciamento del carico.
- Policy di gestione: Permette di definire politiche dettagliate per la gestione delle risorse e delle dipendenze.
- Supporto per vari tipi di risorse: Può gestire servizi, sistemi di file, e qualsiasi risorsa configurabile.
- Casi d'uso: Particolarmente adatto per scenari di alta disponibilità in cui è necessario gestire più tipi di risorse, come database, applicazioni e servizi.
- Richiede una configurazione più complessa rispetto ad altri strumenti, ma offre una maggiore flessibilità e controllo.
- Funziona meglio in combinazione con Corosync per la comunicazione tra i nodi.

Confronto delle soluzioni

Il seguente confronto tra **Ucarp**, **Keepalived**, **Corosync** e **Pacemaker** mette in evidenza le differenze principali tra questi strumenti per la gestione dell’alta disponibilità in ambienti Linux:

1. **Tipo di gestione**:
   - **Ucarp** è focalizzato esclusivamente sul failover delle interfacce di rete, garantendo che l'indirizzo IP virtuale sia sempre disponibile.
   - **Keepalived** aggiunge funzionalità di bilanciamento del carico e health check dei servizi, ideale per configurazioni di rete più avanzate.
   - **Corosync** funge da motore di comunicazione nei cluster, gestendo la membership dei nodi e il quorum.
   - **Pacemaker**, integrato con Corosync, si concentra sulla gestione delle risorse del cluster, offrendo un controllo completo delle dipendenze e del failover.
2. **Semplicità**:
   - **Ucarp** è estremamente semplice da configurare, rendendolo adatto per ambienti meno complessi.
   - **Keepalived** richiede configurazioni intermedie ma rimane più accessibile rispetto a Corosync o Pacemaker.
   - **Corosync** e **Pacemaker** richiedono configurazioni più complesse, ma forniscono un controllo avanzato e maggiore flessibilità.
3. **Funzionalità di health check**:
   - **Ucarp** e **Corosync** offrono funzionalità limitate in questo ambito, focalizzandosi su failover e comunicazione.
   - **Keepalived** include health check nativi, permettendo il monitoraggio attivo dei servizi.
   - **Pacemaker** estende il concetto di health check con policy avanzate per il controllo dello stato delle risorse.
4. **Scalabilità**:
   - **Ucarp** e **Keepalived** sono limitati in termini di scalabilità, essendo orientati a configurazioni semplici o a un numero ristretto di nodi.
   - **Corosync** e **Pacemaker** supportano cluster di grandi dimensioni, risultando ideali per ambienti enterprise con elevate esigenze di scalabilità.
5. **Integrazione**:
   - **Ucarp** offre un'integrazione limitata con altri strumenti.
   - **Keepalived** si integra efficacemente con bilanciatori come HAProxy.
   - **Corosync** e **Pacemaker** funzionano in stretta sinergia, con Corosync come motore di comunicazione e Pacemaker come gestore delle risorse.
6. **Supporto IPv6**:
   - Tutti gli strumenti supportano IPv6, garantendo compatibilità con le reti moderne.

Caratteristica	Ucarp	Keepalived	Corosync	Pacemaker
Tipo di gestione	Failover interfacce di rete	Failover e bilanciamento carico	Comunicazione cluster	Gestione risorse cluster
Semplicità	Alta	Media	Bassa	Bassa
Funzionalità di health check	Limitata	Inclusa	Limitata	Inclusa
Scalabilità	Limitata	Limitata	Alta	Alta
Integrazione	Limitata	Buona (con HAProxy)	Buona (con Pacemaker)	Richiede Corosync
Supporto IPv6	Sì	Sì	Sì	Sì

Questo confronto evidenzia che la scelta dello strumento dipende fortemente dalle esigenze specifiche:

- **Ucarp** e **Keepalived** sono ideali per configurazioni più semplici e reti meno complesse.
- **Corosync** e **Pacemaker** si distinguono per l'affidabilità e la scalabilità in scenari enterprise.

Esempi Applicativi

Ecco un'espansione che include esempi pratici per l'implementazione di **Ucarp**, **Keepalived**, **Corosync** e **Pacemaker** in scenari di alta disponibilità con **Samba**, **NFS**, **MariaDB** e **Nginx**.

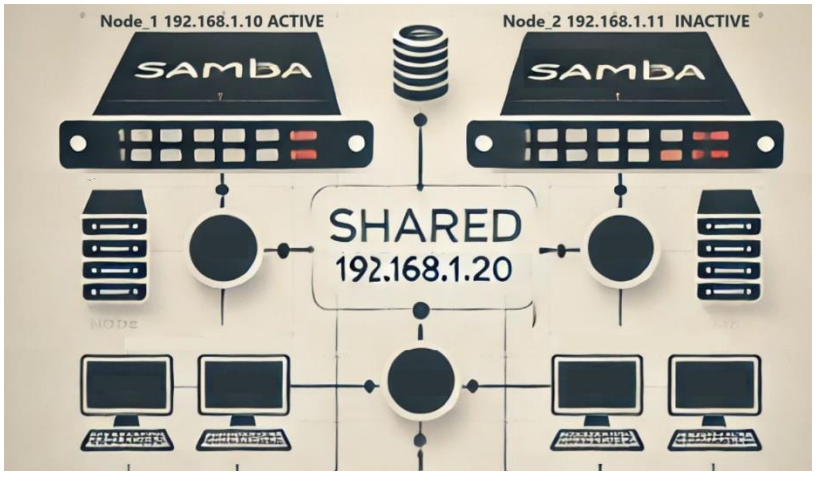
Implementazione di Samba con Ucarp

In questo scenario un server Samba è configurato per fornire condivisioni di file a più client in una rete locale. Per garantire la continuità del servizio, viene utilizzato **Ucarp** per gestire un indirizzo IP virtuale condiviso tra due nodi. Questo approccio assicura che, in caso di guasto del nodo principale, il nodo di riserva possa subentrare immediatamente, mantenendo disponibile il servizio Samba senza interruzioni significative.

La rete include due nodi:

- **Nodo 1 (Attivo):**
  - IP reale: 192.168.1.10
  - IP virtuale: 192.168.1.20
  - Servizio Samba attivo
- **Nodo 2 (Standby):**
  - IP reale: 192.168.1.11
  - IP virtuale: 192.168.1.20

Entrambi i nodi condividono la responsabilità di mantenere l'indirizzo IP virtuale, che viene utilizzato dai client per accedere alle condivisioni Samba.



I passaggi per realizzare questa implementazione sono i seguenti:

1. **Installazione di Ucarp**

   Ucarp deve essere installato su entrambi i nodi. Su una distribuzione Linux basata su Debian, il comando potrebbe essere:

   ```
   sudo apt install ucarp
   ```

2. **Configurazione di Ucarp**

   Configurare **Ucarp** per gestire l'indirizzo IP virtuale su entrambi i nodi. I file di configurazione si trovano in `/etc/ucarp.conf`.

   Configurazione per Node 1:

   ```
   interface eth0
   vip 192.168.1.20
   advertise 1
   password yourpassword
   ```

   Configurazione per Node 2:

   ```
   interface eth0
   vip 192.168.1.20
   advertise 1
   password yourpassword
   ```

3. **Avvio di Ucarp**

   Avviare il servizio Ucarp su entrambi i nodi. Ucarp monitorerà lo stato del nodo attivo (Node 1). In caso di guasto o disconnessione del Node 1, Ucarp farà subentrare automaticamente il Node 2, assegnandogli l'IP virtuale.

4. **Test del failover**

   - Simulare un guasto del Node 1 (ad esempio spegnendolo o disabilitando la rete).
   - Verificare che il Node 2 abbia assunto l'IP virtuale (192.168.1.20) e che i client Samba possano continuare ad accedere alle condivisioni senza interruzioni.

Con questa configurazione, il servizio Samba rimane altamente disponibile grazie al failover automatico gestito da Ucarp. I client possono continuare a utilizzare l'indirizzo IP virtuale per accedere alle condivisioni, indipendentemente dal nodo attualmente attivo.

Implementazione di Nginx con Keepalived

In questo esempio, due server Nginx vengono configurati per garantire l'alta disponibilità e il bilanciamento del carico. Il sistema utilizza **Keepalived**, un software basato sul protocollo VRRP (Virtual Router Redundancy Protocol), per gestire un indirizzo IP virtuale condiviso tra i due server. L'indirizzo IP virtuale consente ai client di accedere al servizio Nginx senza preoccuparsi del server specifico attivo, poiché Keepalived assicura il failover automatico in caso di guasto di uno dei server.

La configurazione prevede due nodi:

- **Node 1 (Attivo):**
  - **IP reale:** 192.168.1.10
  - **IP virtuale:** 192.168.1.20
  - Stato iniziale configurato come **MASTER**.
- **Node 2 (Standby):**
  - **IP reale:** 192.168.1.11
  - **IP virtuale:** 192.168.1.20

- Stato iniziale configurato come **BACKUP**.

L'indirizzo IP virtuale (192.168.1.20) è utilizzato dai client per accedere al bilanciatore Nginx. In caso di guasto del nodo attivo (Node 1), Keepalived consente al nodo standby (Node 2) di assumere il controllo dell'IP virtuale, garantendo la continuità del servizio.

I passaggi per realizzare questa implementazione sono i seguenti:

1. **Installazione di Keepalived**

   Keepalived deve essere installato su entrambi i nodi. Su distribuzioni basate su Debian, il comando è il seguente:

   ```
   sudo apt install keepalived
   ```

2. **Configurazione di Keepalived**

   Ogni nodo richiede un file di configurazione specifico per definire il ruolo e il comportamento rispetto all'indirizzo IP virtuale.

   Configurazione per Node 1 (Attivo): Il file `/etc/keepalived/keepalived.conf` contiene:

   ```
   vrrp_instance VI_1 {
       state MASTER
       interface eth0
       virtual_router_id 51
       priority 100
       advert_int 1
       virtual_ipaddress {
           192.168.1.20
       }
   }
   ```

   Dove:

   - **state MASTER:** Indica che questo nodo è il principale.
   - **priority 100:** La priorità determina quale nodo è preferito come MASTER (valore più alto = maggiore priorità).
   - **virtual_router_id 51:** Identificativo univoco per il gruppo VRRP.

   Configurazione per Node 2 (Standby): Il file `/etc/keepalived/keepalived.conf` contiene:

   ```
   vrrp_instance VI_1 {
       state BACKUP
       interface eth0
       virtual_router_id 51
       priority 90
       advert_int 1
       virtual_ipaddress {
           192.168.1.20
       }
   ```

```
}
```

Dove:

- **state BACKUP:** Indica che questo nodo entrerà in funzione solo se il MASTER non è disponibile.
- **priority 90:** Una priorità inferiore rispetto al nodo MASTER garantisce che questo nodo resti in standby finché il MASTER è operativo.

3. **Configurazione di Nginx**

Entrambi i nodi devono avere un'installazione funzionante di Nginx. La configurazione del bilanciatore di carico (o del server web) dipende dai requisiti specifici, ma è essenziale che entrambi i nodi siano configurati per gestire lo stesso traffico.

Installare Nginx:

```
sudo apt install nginx
```

Copiare i file di configurazione Nginx su entrambi i nodi per garantire coerenza nei servizi esposti.

4. **Avvio dei servizi**

- Avviare Nginx su entrambi i nodi:

  ```
  sudo systemctl start nginx
  ```

- Avviare Keepalived:

  ```
  sudo systemctl start keepalived
  ```

5. **Test del failover**

- Spegnere o disabilitare temporaneamente la rete su Node 1.
- Verificare che Node 2 abbia assunto il controllo dell'indirizzo IP virtuale (192.168.1.20). Ciò può essere verificato utilizzando ip addr su Node 2.
- Ripristinare Node 1 e osservare che riprende il ruolo di MASTER.

Grazie all'utilizzo di Keepalived, il sistema garantisce che l'indirizzo IP virtuale sia sempre disponibile, indipendentemente dallo stato dei singoli nodi. I client continuano a connettersi al servizio Nginx senza interruzioni, migliorando l'affidabilità complessiva del sistema. Questo approccio è ideale per scenari web in cui il bilanciamento del carico e l'alta disponibilità sono fondamentali.

Implementazione di MariaDB con Corosync e Pacemaker

In questo esempio, due server MariaDB vengono configurati in un cluster per garantire l'alta disponibilità dei database. Questo approccio consente di mantenere l'accesso continuo al database, anche in caso di guasto di uno dei nodi. La configurazione utilizza **Corosync** come sistema di comunicazione tra i nodi e **Pacemaker** per la gestione delle risorse del cluster.

La configurazione prevede:

- **Node 1 (Attivo):** Questo nodo inizialmente ospita la risorsa MariaDB ed è configurato per servire richieste client.
- **Node 2 (Standby):** Questo nodo rimane in attesa, pronto a subentrare in caso di guasto del Node 1.

L'alta disponibilità è ottenuta assicurando che solo un nodo alla volta gestisca il servizio MariaDB, mentre l'altro agisce come riserva.

I passaggi per realizzare questa implementazione sono i seguenti:

1. **Installazione di Corosync e Pacemaker**

   Su entrambi i nodi, installare i pacchetti richiesti:

   ```
   sudo apt install corosync pacemaker pcs
   ```

   Abilitare e avviare i servizi:

   ```
   sudo systemctl enable corosync pacemaker
   sudo systemctl start corosync pacemaker
   ```

   Configurare un account amministrativo per **pcs** (ad esempio, `hacluster`) e sincronizzarlo su entrambi i nodi:

   ```
   sudo passwd hacluster
   ```

2. **Configurazione di Corosync**

Il file di configurazione di Corosync si trova in `/etc/corosync/corosync.conf`. Personalizzare il file come segue:

```
totem {
    version: 2
    secauth: off
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.1.0
        mcastaddr: 226.94.1.1
        mcastport: 5405
    }
}

nodelist {
    node {
        ring0_addr: node1
        nodeid: 1
    }
    node {
        ring0_addr: node2
        nodeid: 2
    }
}
```

Questa configurazione definisce:

- **bindnetaddr:** L'indirizzo di rete comune ai nodi del cluster.
- **nodelist:** L'elenco dei nodi, con indirizzi IP e ID univoci.

Dopo aver configurato il file, riavviare Corosync:

```
sudo systemctl restart corosync
```

3. **Creazione del cluster con Pacemaker**

Creare il cluster con **pcs**:

```
sudo pcs cluster setup --name mariadb_cluster node1 node2
sudo pcs cluster start --all
```

Verificare lo stato del cluster:

```
sudo pcs status
```

4. **Configurazione della risorsa MariaDB**

Aggiungere MariaDB come risorsa nel cluster:

```
sudo pcs resource create mariadb ocf:heartbeat:mysql \
    config=/etc/my.cnf \
    op start timeout=60s \
    op stop timeout=60s \
    op monitor interval=20s
```

Questa configurazione:

- Specifica il percorso del file di configurazione di MariaDB (/etc/my.cnf).
- Definisce operazioni di avvio, arresto e monitoraggio con relativi timeout.

Impostare un vincolo per garantire che MariaDB venga eseguito su un solo nodo alla volta:

```
sudo pcs constraint colocation add mariadb with master
sudo pcs constraint order start mariadb then mysql
```

5. **Monitoraggio e test del failover**

Verificare che MariaDB sia attivo su Node 1:

```
sudo pcs resource show
```

Simulare un guasto del nodo attivo:

- Arrestare Pacemaker su Node 1:

  ```
  sudo systemctl stop pacemaker
  ```

- Verificare che Node 2 abbia assunto il controllo del servizio:

```
sudo pcs status
```

Con questa configurazione, il servizio MariaDB rimane disponibile anche in caso di guasto di uno dei nodi. **Corosync** garantisce la comunicazione tra i nodi, mentre **Pacemaker** gestisce il failover e il bilanciamento delle risorse. Questo approccio è ideale per database critici che richiedono continuità operativa.

Implementazione di NFS con Pacemaker e Corosync

In questo esempio, due server NFS vengono configurati in un cluster ad alta disponibilità utilizzando **Pacemaker** e **Corosync**. Questo approccio garantisce che il servizio NFS rimanga sempre disponibile per i client, anche in caso di guasto di uno dei nodi. I due server agiscono in modalità attivo/standby: uno serve i file condivisi mentre l'altro è pronto a subentrare in caso di necessità.

- **Node 1 (Attivo):** Serve il servizio NFS e ospita i file condivisi.
- **Node 2 (Standby):** Resta in attesa e subentra automaticamente in caso di guasto del Node 1.

L'architettura garantisce che il servizio NFS sia attivo su un solo nodo alla volta per evitare conflitti di accesso ai file.

I passaggi per realizzare questa implementazione sono i seguenti:

1. **Installazione di Corosync e Pacemaker**

Installare i pacchetti richiesti su entrambi i nodi:

```
sudo apt install corosync pacemaker pcs nfs-kernel-server
```

Abilitare e avviare i servizi:

```
sudo systemctl enable corosync pacemaker
sudo systemctl start corosync pacemaker
```

2. **Configurazione di Corosync**

Seguire i passaggi descritti in precedenza per configurare Corosync, impostando i dettagli di rete per consentire la comunicazione tra i nodi.

3. **Creazione della risorsa NFS**

Creare un'unità condivisa per i file NFS sul file system di entrambi i nodi (es. `/mnt/nfs`). Assicurarsi che la directory condivisa sia disponibile su entrambi i nodi e configurare il servizio NFS:

a) Modificare il file di configurazione di NFS:

```
sudo nano /etc/exports
```

Aggiungere la directory condivisa e i permessi di accesso:

```
/mnt/nfs *(rw,sync,no_root_squash)
```

b) Esportare la configurazione:

```
sudo exportfs -a
```

c) Creare una risorsa NFS in Pacemaker:

```
sudo pcs resource create nfs-server nfsserver nfs_shared_infodir=/mnt/nfs
\op start timeout=60 op stop timeout=60 op monitor interval=30
```

4. **Aggiunta di vincoli di colocation**

Per garantire che NFS venga eseguito su un solo nodo alla volta, configurare un vincolo di colocation:

```
sudo pcs constraint colocation add nfs-server with node1
```

Questa configurazione assicura che NFS sia attivo solo sul nodo designato, evitando conflitti di accesso simultaneo.

5. **Monitoraggio e test del failover**

Verificare lo stato delle risorse per assicurarsi che il servizio NFS sia attivo sul nodo previsto:

```
sudo pcs status
```

Simulare un guasto del nodo attivo arrestando Pacemaker:

```
sudo systemctl stop pacemaker
```

Verificare che il nodo standby abbia assunto il controllo del servizio:

```
sudo pcs status
```

Con questa configurazione, il servizio NFS è sempre disponibile per i client, anche in caso di guasto di uno dei nodi. **Corosync** garantisce la comunicazione affidabile tra i nodi, mentre **Pacemaker** gestisce il failover automatico e assicura che il servizio venga eseguito su un solo nodo alla volta. Questo approccio è ideale per ambienti in cui l'accesso continuo e condiviso ai file è essenziale.

Soluzioni a confronto

La scelta tra Ucarp, Keepalived, Corosync e Pacemaker dipende fortemente dai requisiti specifici dell'ambiente in cui si intende implementare la soluzione. **Ucarp** è più adatto per ambienti semplici e limitati, mentre **Keepalived** si distingue per bilanciamento del carico e failover. **Corosync** e **Pacemaker** sono più adatti per scenari complessi di clustering e gestione delle risorse. La combinazione di Corosync con Pacemaker offre una soluzione robusta per ambienti enterprise che necessitano di alta disponibilità e resilienza. L'implementazione di alta disponibilità può variare notevolmente a seconda delle esigenze specifiche. **Ucarp** è adatto per configurazioni semplici, mentre **Keepalived** è più flessibile per il bilanciamento del carico con Nginx. L'abbinamento di **Corosync** e **Pacemaker** fornisce una gestione robusta e complessa per database come **MariaDB** e file sharing come **NFS**.

Implementazione di Router OpenWRT in Alta Disponibilità

Nelle reti moderne, l'affidabilità della connessione a Internet è essenziale, specialmente in ambienti in cui il guasto di un router potrebbe causare interruzioni significative del servizio. Configurare due router OpenWRT in modalità **alta disponibilità (HA)** consente di superare questa criticità. In questa configurazione, un router opera come **attivo**, gestendo il traffico di rete, mentre l'altro rimane in **standby**, pronto a subentrare automaticamente in caso di guasto. Questo risultato si ottiene utilizzando **Keepalived** per gestire un indirizzo IP virtuale condiviso e, opzionalmente, **conntrack-tools** per sincronizzare le connessioni NAT tra i due router.

L'obiettivo di questa implementazione è garantire la **continuità del servizio** attraverso:

1. **Failover trasparente**: Quando il router attivo subisce un guasto, il router di backup assume automaticamente il controllo del traffico di rete.
2. **Indirizzo IP virtuale condiviso**: Gli host sulla LAN utilizzano un unico gateway IP per accedere a Internet, indipendentemente dal router attualmente in uso.
3. **Semplicità e compatibilità**: La configurazione si basa su strumenti consolidati e configurazioni standard di OpenWRT, adattabili a reti casalinghe o aziendali.

Architettura della Soluzione

- **Router 1 (Attivo)**: Gestisce il traffico LAN e WAN durante il normale funzionamento.
- **Router 2 (Backup)**: Rimane in standby, monitorando lo stato del router attivo. Subentra automaticamente in caso di guasto.
- **Indirizzo IP virtuale (VIP)**: Gli host LAN utilizzano l'IP virtuale come gateway. Il VIP viene gestito dinamicamente da Keepalived per garantire l'accesso ininterrotto alla rete.

Configurazione dei Router

1. Configurazione del Router Attivo (Router 1)

Il primo router deve essere configurato come attivo. Utilizzeremo gli indirizzi IP seguenti:

- **IP LAN interno**: 192.168.1.2/24. Questo indirizzo consente al router di comunicare con i dispositivi della rete locale.
- **IP WAN**: 192.168.0.2/24. Questo indirizzo viene utilizzato per accedere alla rete del provider (usando un doppio NAT o DMZ).
- **Gateway WAN**: 192.168.0.1. Questo rappresenta l'indirizzo del router del provider.

Configurare il DHCP con le impostazioni di default. Successivamente, sarà necessario adattarlo per supportare l'indirizzo IP virtuale condiviso.

2. Configurazione del Router di Backup (Router 2)

Il router di backup deve essere configurato per entrare in funzione solo quando il router attivo non è più disponibile:

- **IP LAN interno**: 192.168.1.3/24. È importante che questo indirizzo non entri in conflitto con l'IP del router attivo.
- **IP WAN**: 192.168.0.3/24. Questo indirizzo rappresenta la connessione del router alla rete del provider.

- **Gateway WAN**: Identico al primo router, 192.168.0.1.

Durante la configurazione iniziale, verificare che entrambi i router possano comunicare con i dispositivi sulla LAN e che utilizzino lo stesso schema di routing.

Configurazione di Keepalived

Keepalived è un demone utilizzato per implementare il protocollo VRRP (Virtual Router Redundancy Protocol). Questo consente ai router di condividere dinamicamente un indirizzo IP virtuale.

Su entrambi i router occorre lanciare il seguente comando per installare Keepalived:

```
opkg update && opkg install keepalived
```

Keepalived richiede un file di configurazione per definire il comportamento di failover. Creare il file `/etc/keepalived/keepalived.conf` su entrambi i router.

Esempio di configurazione per router simmetrici:

```
vrrp_sync_group G1 {
  group {
    E1
    I1
  }
}

vrrp_instance I1 {
  state backup
  interface br-lan
  virtual_router_id 51
  priority 101
  advert_int 1
  virtual_ipaddress {
    192.168.1.4/24
  }
  authentication {
    auth_type PASS
    auth_pass s3cret
  }
  nopreempt
}

vrrp_instance E1 {
  state backup
  interface eth0.2
  virtual_router_id 51
  priority 101
  advert_int 1
  virtual_ipaddress {
    192.168.0.4/24
  }
  virtual_routes {
    src 192.168.0.4 to 0.0.0.0/0 via 192.168.0.1 dev eth0.2 metric 5
  }
  authentication {
    auth_type PASS
```

```
        auth_pass s3cret
      }
      nopreempt
    }
```

Aggiungere una configurazione in `/etc/config/keepalived` per garantire l'avvio automatico:

```
config global_defs 'globals'
   option alt_config_file "/etc/keepalived/keepalived.conf"
```

Configurazione di conntrackd (opzionale)

Conntrack-tools sincronizza le connessioni NAT tra i router. È particolarmente utile per garantire che le connessioni attive non vengano interrotte durante il failover.

Esempio di configurazione per `/etc/conntrackd/conntrackd.conf`:

```
Sync {
    Mode FTFW {
        CommitTimeout 1800
        PurgeTimeout 5
    }
    UDP {
        IPv4_address "192.168.0.2"
        IPv4_Destination_Address "192.168.0.3"
        Port 3780
    }
}

General {
    HashSize 32768
    LogFile on
    Syslog on
    NetlinkBufferSize 2097152
}
```

Configurazione DHCP

Aggiornare il file di configurazione di `dnsmasq` per assegnare l'indirizzo IP virtuale come gateway:

```
config dhcp 'lan'
    option force '1'
    list dhcp_option '3,192.168.1.4'
    list dhcp_option '6,192.168.1.4'
```

Test della configurazione

1. **Simulare un guasto**: Spegnere il router attivo o scollegare il cavo WAN.
2. **Monitorare il failover**: Verificare che il router di backup assuma l'indirizzo IP virtuale e funzioni come gateway.
3. **Controllare la connessione**: Assicurarsi che gli host LAN possano accedere a Internet senza interruzioni.

Conclusioni

Questa configurazione garantisce un gateway LAN altamente disponibile. Keepalived gestisce il failover dell'indirizzo IP virtuale, mentre conntrackd (se utilizzato) sincronizza le connessioni NAT. La soluzione è scalabile, robusta e adatta a reti in cui l'accesso ininterrotto è una priorità.

Bibliografia

1. Wikipedia. **Common Address Redundancy Protocol.** Disponibile online: https://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol , ultima consultazione il 10 Gennaio 2025.
2. Linux Virtual Server Knowledge Base. **UCARP**. Disponibile online: https://kb.linuxvirtualserver.org/wiki/UCARP, ultima consultazione il 10 Gennaio 2025.
3. Rapscallion. **High Availability VIP Management With ucarp**. Disponibile online: https://rascaldev.io/2018/03/13/vip-management-with-ucarp/, ultima consultazione il 10 Gennaio 2025.
4. Sebest's Nuage. **IP failover with Ucarp on Ubuntu**. Disponibile online: https://sebest.github.io/post/ip-failover-with-ucarp-on-ubuntu/ , ultima consultazione il 10 Gennaio 2025.
5. GitHub. UCARP. Disponibile online: https://github.com/hardfalcon/ucarp, ultima consultazione il 10 Gennaio 2025.
6. SysTutorials. **keepalived-Linux Manuals**. Disponibile online: https://www.systutorials.com/docs/linux/man/8-keepalived/, ultima consultazione il 10 Gennaio 2025.
7. IBM documentation. **Keepalived and HAProxy**. Disponibile online: https://www.ibm.com/docs/en/solution-assurance?topic=available-keepalived-haproxy , ultima consultazione il 10 Gennaio 2025.
8. Red Hat documentation. **Keepalived Overview**. Disponibile online: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/load_balancer_administration/ch-keepalived-overview-vsa, ultima consultazione il 10 Gennaio 2025.
9. Wikipedia. **Corosync Cluster Engine.** Disponibile online: https://en.wikipedia.org/wiki/Corosync_Cluster_Engine, ultima consultazione il 10 Gennaio 2025.
10. Red Hat Customer Portal. **Exploring RHEL High Availability's Components – corosync**. Disponibile online: https://access.redhat.com/articles/2661941, ultima consultazione il 10 Gennaio 2025.
11. Quoll tech. **Cluster Linux in test con Pacemaker, Corosync e pcs**. Disponibile online: https://quoll.it/cluster-linux-in-test-con-pacemaker-corosync-e-pcs/, ultima consultazione il 10 Gennaio 2025.
12. Linux Console. **Configura l'alta disponibilità con Corosync e Pacemaker**. Disponibile online: https://it.linux-console.net/?p=27967#google_vignette, ultima consultazione il 10 Gennaio 2025.
13. GitHub. **Corosync**. Disponibile online: https://github.com/corosync/corosync/wiki, ultima consultazione il 10 Gennaio 2025.
14. IBM documentation. **Pacemaker (Linux)**. Disponibile online: https://www.ibm.com/docs/it/db2/11.5?topic=software-pacemaker-linux, ultima consultazione il 10 Gennaio 2025.
15. ClusterLabs. **Pacemaker Administration**. Disponibile online: https://clusterlabs.org/projects/pacemaker/doc/2.1/Pacemaker_Administration/singlehtml/, ultima consultazione il 10 Gennaio 2025.
16. Wikipedia. **Pacemaker**. Disponibile online: https://en.wikipedia.org/wiki/Pacemaker_%28software%29, ultima consultazione il 10 Gennaio 2025.