Documentación de Arquitectura: Clúster Kubernetes Híbrido

By Luis Antonio Calvo Quispe

23 de noviembre de 2025

Índice

1. Resumen General

Este documento detalla la arquitectura de un clúster de Kubernetes auto-alojado (on-premise) diseñado para alta disponibilidad, flexibilidad y exposición segura de servicios a Internet. La arquitectura se caracteriza por su naturaleza híbrida, combinando nodos de diferentes arquitecturas (x86_64 y ARM64), un sistema de almacenamiento centralizado basado en NFS, y un doble esquema de balanceo de carga para el plano de control y los servicios.

2. Arquitectura de Red

La red es un pilar fundamental de este clúster, dividida en dos sistemas de balanceo de carga completamente independientes, cada uno con un propósito distinto:

- **Balanceo del Plano de Control (Keepalived + HAProxy):** Este sistema se encarga exclusivamente de garantizar la alta disponibilidad del **API Server de Kubernetes**. Funciona a nivel de TCP (Capa 4) y opera de forma externa a la lógica de Kubernetes. Es esencial tener un punto de acceso único y robusto para que los nodos y los administradores puedan comunicarse con el clúster en todo momento. No se puede usar un balanceador interno de Kubernetes para esta tarea, ya que estos dependen de que el API Server ya esté funcionando.

  Analisis de Impacto: Se perdería la capacidad de **administrar el clúster** (los comandos `kubectl` fallarían) y se detendrían las tareas de auto-reparación. Sin embargo, es crucial entender que las **aplicaciones en ejecución no se verían afectadas** a corto plazo y seguirían atendiendo tráfico, ya que el plano de datos es independiente.

- **Balanceo de Servicios (MetalLB):** Este sistema está integrado con Kubernetes y su función es exponer los **servicios y aplicaciones** que se ejecutan dentro del clúster a la red local. Cuando se crea un servicio de tipo `LoadBalancer`, MetalLB le asigna una IP externa de su propio rango. Este balanceador gestiona el tráfico de las aplicaciones, no el tráfico administrativo del clúster.

  Analisis de Impacto: Un fallo en MetalLB impediría que se anuncien las IPs de los servicios en la red. Esto provocaría la **pérdida de acceso externo a todas las aplicaciones** a través del Ingress Controller, ya que su IP se volvería inalcanzable. A diferencia del caso anterior, la **administración del clúster (`kubectl`) seguiría funcionando** con normalidad, permitiendo diagnosticar y resolver el problema sin afectar la gestión del sistema.

- **Exposición a Internet (FRP):** Dado que el clúster es privado, se utiliza un cliente **FRP (Fast Reverse Proxy)** para exponer los servicios a Internet. Este componente establece un túnel reverso con un servidor FRP en un VPS público y redirecciona el tráfico externo (ej. puertos 80 y 443) desde el VPS hacia la IP del Ingress Controller (`192.168.100.240`), haciendo accesibles las aplicaciones desde fuera de la red local.

  Analisis de Impacto: Un fallo en el sistema FRP (ya sea en el cliente o en el servidor) cortaría el túnel reverso. Esto impediría únicamente el **acceso a las aplicaciones desde Internet**. El clúster seguiría funcionando y siendo accesible tanto para la **administración** como para los **usuarios de la red local**.

Esta separación es fundamental: uno asegura la estabilidad y el acceso al cerebro del clúster (el plano de control), mientras que el otro se encarga de dar acceso a las aplicaciones que este orquesta.

IP	Componente Principal	Función
192.168.100.230	Keepalived + HAProxy	VIP para el plano de control de Kubernetes (API Server).
192.168.100.240	MetalLB + Nginx Ingress	IP externa para servicios expuestos (Ingress).

Cuadro 1: Tabla Comparativa de IPs de Balanceo de Carga.

A continuación, se detallan los componentes de cada sistema.

2.1. Plano de Control y Alta Disponibilidad

Para garantizar la disponibilidad del API Server de Kubernetes, se utiliza una combinación de **Keepalived** y **HAProxy**.

- **Keepalived:** Gestiona una Dirección IP Virtual (VIP) **192.168.100.230**. Esta IP actúa como el punto de entrada flotante para el plano de control. El nodo Maestro del balanceador es `n100-004`, con los servidores NAS `(nas-001,nas-002,...)` actuando como respaldo.
- **HAProxy:** Se ejecuta en los nodos del balanceador y distribuye el tráfico TCP del puerto 6443 (API de Kubernetes) entre los tres nodos Maestros del clúster: `n100-001`, `n100-002` y `n100-003`.
- **Estadísticas de HAProxy:** El estado del balanceador y sus backends se puede consultar en http://161.132.4.98:8404/stats

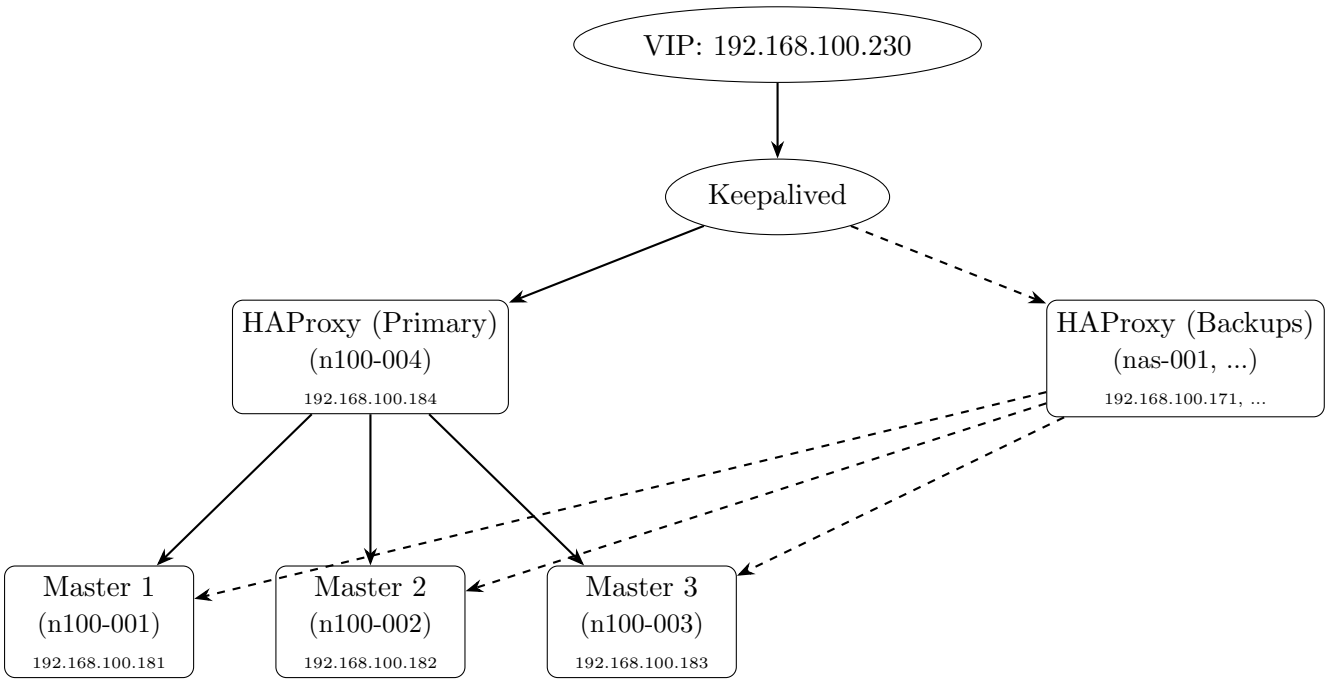

Figura 1: Diagrama del Plano de Control con Alta Disponibilidad.

Al inicializar el clúster con `kubeadm`, se especifica esta VIP como el endpoint del plano de control:

2.2. Exposición de Servicios y Tráfico de Entrada (Ingress)

La exposición de servicios a la red local y a Internet se gestiona de la siguiente manera:

- **MetalLB:** Actúa como balanceador de carga de red para el clúster. Proporciona direcciones IP externas a servicios de tipo `LoadBalancer`. Está configurado en modo Layer 2 para anunciar IPs desde el rango **192.168.100.240-192.168.100.254**.
- **Nginx Ingress Controller:** Es el principal punto de entrada para el tráfico HTTP/S. Su servicio se expone a la red local mediante MetalLB, que le asigna la IP **192.168.100.240**.

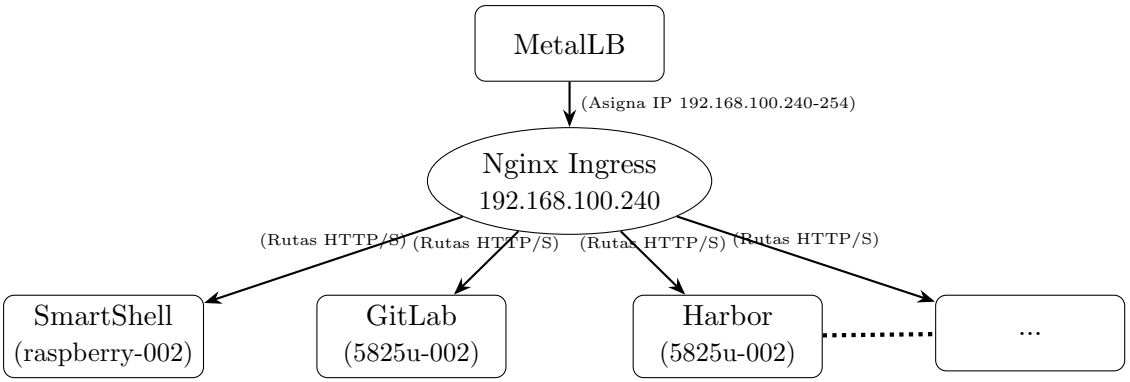

Figura 2: Diagrama de Ingress y MetalLB.

2.3. FRP (Fast Reverse Proxy)

FRP sirve para la exposición de servicios a Internet. Funciona mediante un par de aplicaciones: un servidor (`frps`) en una máquina pública y un cliente (`frpc`) dentro de la red local.

2.3.1. Servidor FRP (frps)

El servidor FRP se ejecuta en un VPS con una IP pública estática (`161.132.4.98`).

- **Función:** Actúa como punto de encuentro para los clientes FRP, recibiendo conexiones de ellos y reenviando el tráfico público entrante a través de los túneles establecidos.
- **Panel de Control:** Ofrece un panel de control web para monitorear el estado de los clientes y los túneles, accesible en `http://161.132.4.98:7500/`.

2.3.2. Cliente FRP (frpc)

El cliente FRP se despliega dentro del clúster de Kubernetes.

- **Función:** Establece una conexión persistente (túnel) con el servidor `frps`.
- **Redirección de Tráfico:** Su configuración principal consiste en reenviar el tráfico recibido en los puertos públicos 80 y 443 del servidor `frps` hacia la IP del Ingress Controller de Nginx dentro del clúster (`192.168.100.240`). De esta manera, las peticiones que llegan a la IP pública son dirigidas de forma segura a las aplicaciones que se ejecutan en Kubernetes.

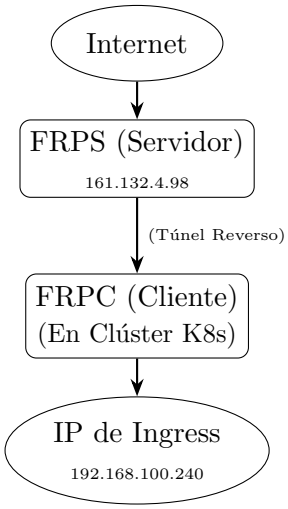

Figura 3: Diagrama del Túnel FRP.

3. Arquitectura General de Tráfico y Balanceo

La arquitectura del clúster se basa en dos sistemas de balanceo de carga independientes que atienden a dos propósitos distintos: uno para la **gestión del clúster** (plano de control) y otro para la **exposición de aplicaciones** (plano de datos). El siguiente diagrama unifica los flujos de tráfico para proporcionar una visión general, integrando los componentes de red en una sola vista.

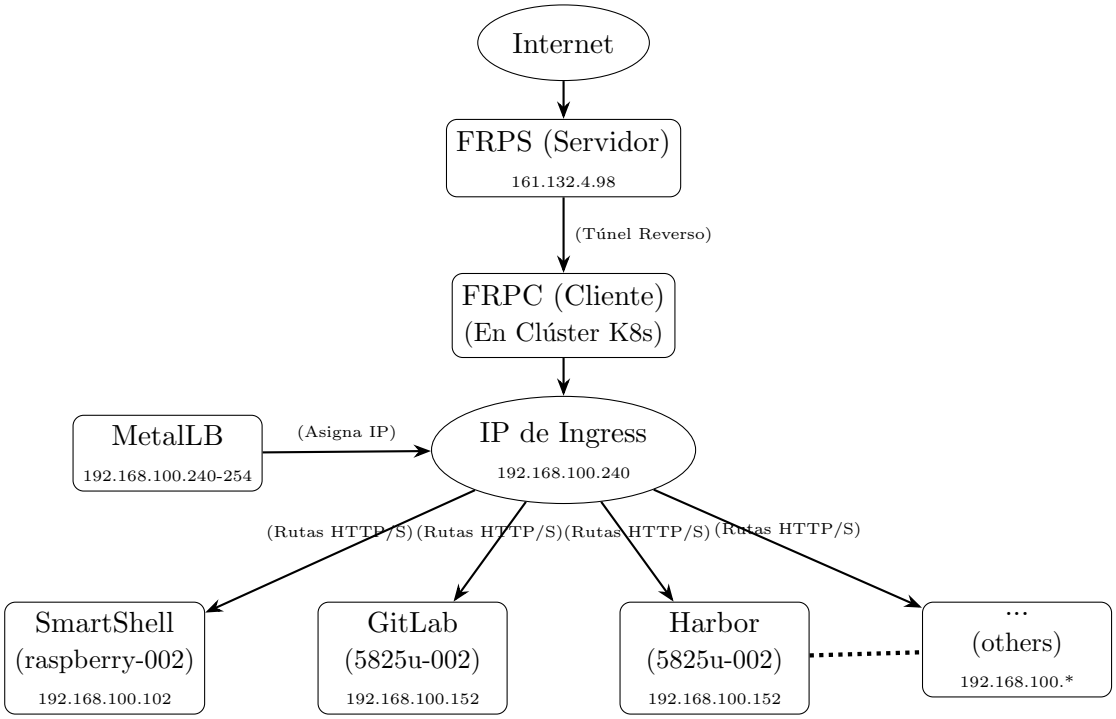

Figura 4: Diagrama Integrado de Flujo de Tráfico y Balanceo de Carga.

4. Nodos del Clúster

El clúster es heterogéneo, compuesto por nodos de diferentes arquitecturas para optimizar el consumo de energía y el rendimiento según la carga de trabajo.

4.1. Nodos Maestros (Control Plane)

El plano de control reside en tres nodos idénticos para garantizar quórum y alta disponibilidad.

- **Hosts:** `n100-001`, `n100-002`, `n100-003`.
- **Arquitectura:** x86_64 (basados en CPUs Intel N100).
- **Rol:** Ejecutan los componentes críticos de Kubernetes como `etcd`, `kube-apiserver`, `kube-scheduler` y `kube-controller-manager`.

4.2. Nodos de Trabajo (Workers)

Los nodos de trabajo son una mezcla de arquitecturas ARM64 y x86_64.

- **Hosts ARM64:** Múltiples Raspberry Pi (`raspberry-001` a `raspberry-008`). Ideales para cargas de trabajo ligeras y de bajo consumo. Su arquitectura es `aarch64`.

- **Hosts x86_64:** Nodos adicionales (`5825u-001`, etc.) para cargas de trabajo que requieren la arquitectura amd64.

El uso de arquitecturas mixtas requiere la creación de imágenes de contenedor multi-arquitectura para asegurar que las aplicaciones puedan ser desplegadas en cualquier nodo.

```
NAME            STATUS                ROLES           AGE   VERSION
5825u-001       Ready                 worker          27d   v1.32.3
5825u-002       Ready                 worker          27d   v1.32.3
n100-001        Ready                 control-plane   98d   v1.32.3
n100-002        Ready                 control-plane   98d   v1.32.3
n100-003        Ready                 control-plane   98d   v1.32.3
raspberry-001   Ready                 worker          98d   v1.32.3
raspberry-002   Ready                 worker          98d   v1.32.3
raspberry-003   Ready                 worker          98d   v1.32.3
raspberry-004   Ready                 worker          98d   v1.32.3
raspberry-005   Ready                 worker          98d   v1.32.3
raspberry-006   Ready                 worker          98d   v1.32.3
raspberry-007   Ready                 worker          98d   v1.32.3
raspberry-008   Ready                 worker          98d   v1.32.3
```

5. Arquitectura de Almacenamiento

El almacenamiento persistente se centraliza utilizando un servidor NFS dedicado, al cual el clúster se conecta a través del driver CSI (Container Storage Interface) para NFS.

- **Servidor NFS:** El host `nas-001` (IP: `192.168.100.171`) actúa como el servidor principal de NFS.
- **Provisionador CSI:** El clúster utiliza `nfs.csi.k8s.io` para permitir que los volúmenes persistentes (Persistent Volumes) se monten desde el servidor NFS.
- **StorageClasses:** Se han definido múltiples clases de almacenamiento (`nas-001`, `nas-002`, `nas-003`) que apuntan al mismo servidor NFS, probablemente a diferentes directorios compartidos (*shares*), permitiendo diferenciar políticas de almacenamiento.

A continuación un ejemplo de la definición de una `StorageClass`:

```
apiversion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nas-001
provisioner: nfs.csi.k8s.io
parameters:
  server: 192.168.100.171
  share: /mnt/server
reclaimPolicy: Delete
```

6. TLS y Certificados

La gestión de certificados TLS para los servicios expuestos a través de Ingress está automatizada mediante **Cert-Manager**.

- **Cert-Manager:** Es un controlador de Kubernetes que solicita y renueva automáticamente certificados TLS de diversas fuentes.

- **Let's Encrypt:** Se utiliza como la Autoridad de Certificación (CA) para emitir certificados gratuitos y confiables para los dominios públicos. Cert-Manager se comunica con Let's Encrypt para validar la propiedad del dominio y obtener los certificados para las reglas de Ingress que los soliciten.

7. Diagrama de Arquitectura

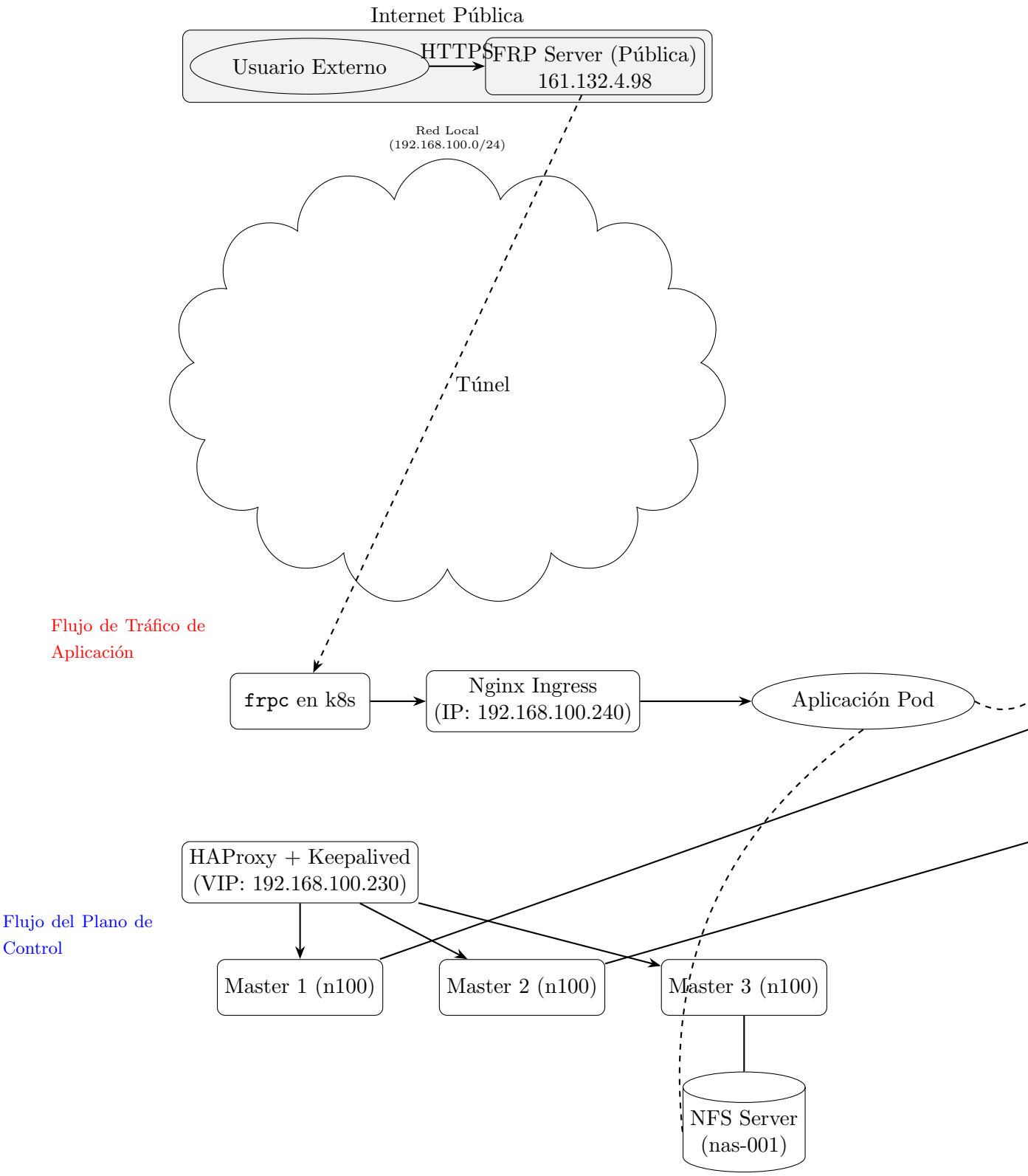

Figura 5: Diagrama de la arquitectura del clúster de Kubernetes.