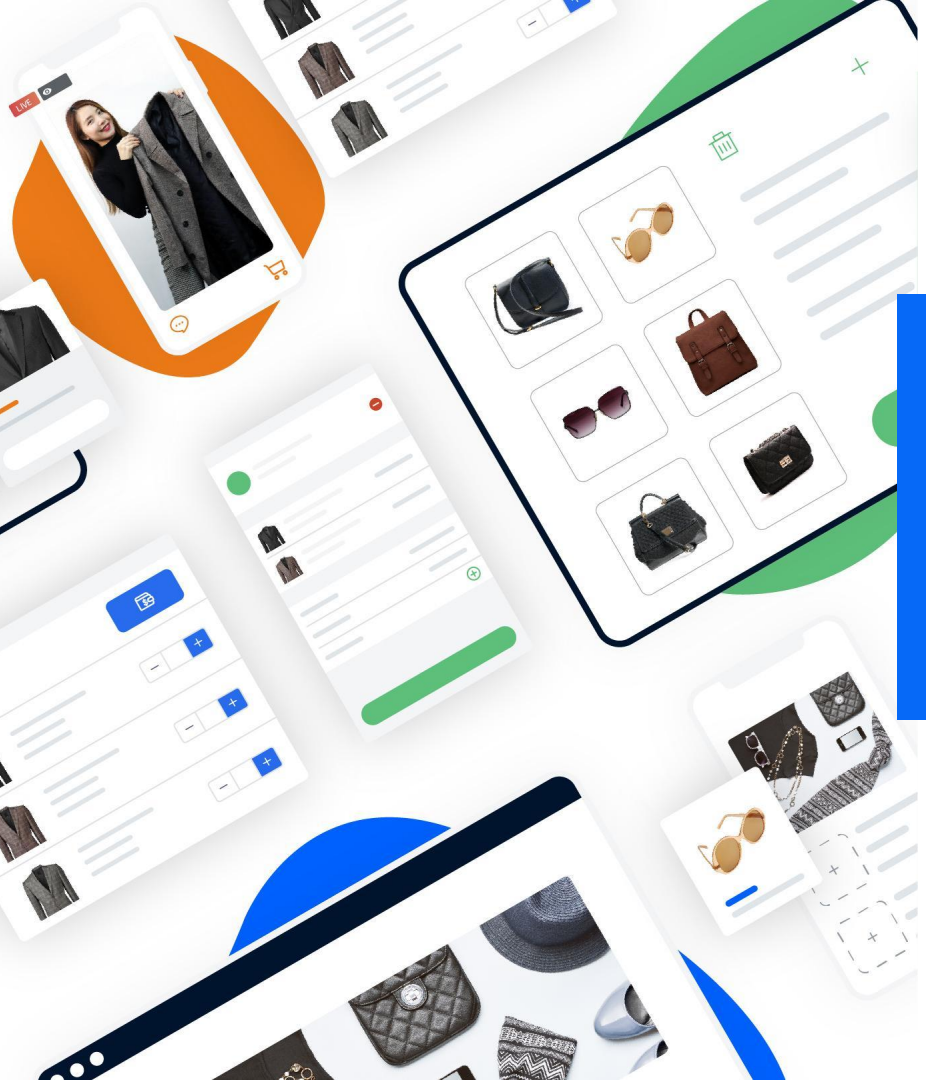


從零打造 eBPF CNI Plugin：揭秘 Cilium 封包處理核心原理

Shiun Chiu

SHOPLINE

About me

- 在 SHOPLINE 的 Cloud Team (Platform) 擔任 Cloud Engineer
- 喜歡鑽研 Serverless, Container, Networking… 等技術
- 業餘身份
  - AWS Community Builder
  - AWS Educate Cloud Ambassador

開場序：靈感來源

靈感來自 Liz Rice - Containers From Scratch • Liz Rice • GOTO 2018

https://youtu.be/8fi7uSYlOdc?si=oBn0lXJ9mjF7hxKA

開場序

- 團隊近期在導入 NetworkPolicy
- 團隊在 Networking 領域有一些藍圖想實現，Cilium 提供了很多進階方案
- 這場演講全程都是技術內容，撐下去，我保證收穫滿滿
- 本演講預設**聽眾已對 K8s 與 Linux Networking 有基礎**

2024 K8s Summit - SHOPLINE

低成本加強 Service 網路可視化：零停機切換到 Cilium CNI 經驗分享

https://k8s.ithome.com.tw/2024/session-page/3282

SHOPLINE

你是否也遇過？

- **尚未導入 Cilium 的你**
  - iptables-based Network Policy 效能慢正在考慮新解決方案，例如 Cilium
- **已經導入 Cilium 的你**
  - 有人問你 Cilium 是怎麼控制封包的？但只能回答出很抽象的概念
  - 新同事對 Cilium 不熟，但 Cilium 學習曲線很高，想學習卻不知如何下手
  - 封包不知道跑哪去了？
  - NetworkPolicy 怎麼不如預期，Cilium 到底怎麼決策封包 Allow/Deny？

從零打造  eBPF CNI Plugin

揭秘  cilium 封包處理核心原理

從零打造 eBPF CNI Plugin

揭秘  cilium 封包處理核心原理

動手造馬車 (eBPF CNI Plugin)

來理解 法拉利 (Cilium) 的核心運作

議程

- 第一章、動手寫基礎 CNI Plugin
- 第二章、利用 eBPF 賦予 Kernel 可程式化的能力
- 第三章、在 eBPF 實現 Identity-based 封包決策
- Q&A

目前環境

```
$ kubeadm init
```

- K8s 1.32
- Single Node Cluster
- 尚未安裝 CNI Plugin（我們自己寫）

Node (**NotReady**)

目前環境

```
$ kubectl run my-pod
```

- K8s 1.32
- Single Node Cluster
- 尚未安裝 CNI Plugin（我們自己寫）

Node (**NotReady**)

Pending...

來去 Terminal 看看環境

然後起一個 Pod 看會怎樣

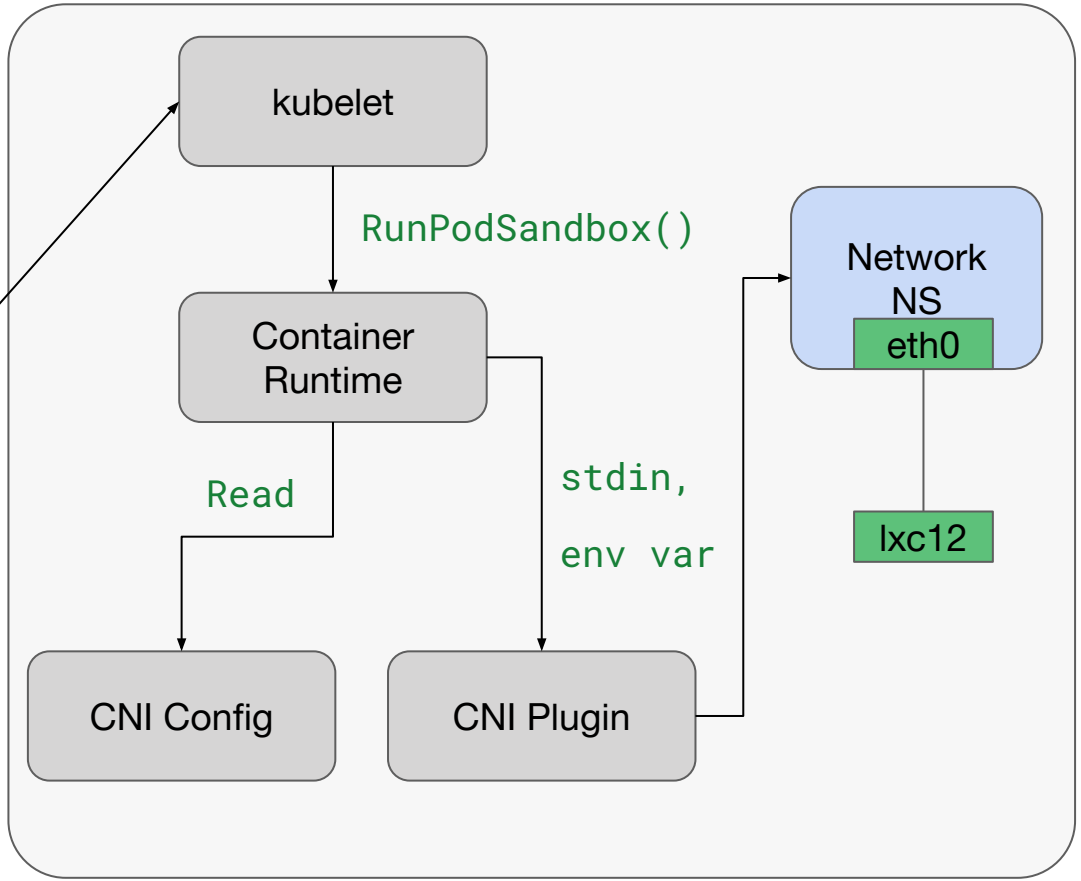
先備知識 - CNI

containernetworking/cni

- 是一個**規範 (Spec)**
- 依照此 Spec 實作出來的 binary 檔案就是 **CNI Plugin**
- **標準化 Container runtime 和 CNI Plugin 的溝通**
  - Plugin 必須能理解 `CNI_COMMAND` (例如：ADD / DEL)
  - Plugin 接收 **stdin JSON config + environment variables**
  - Plugin 最後輸出 **stdout JSON** 給 container runtime
- **推薦影片：Tutorial: From CNI Zero to CNI Hero: A Kubernetes Networking Tutorial Using CNI**

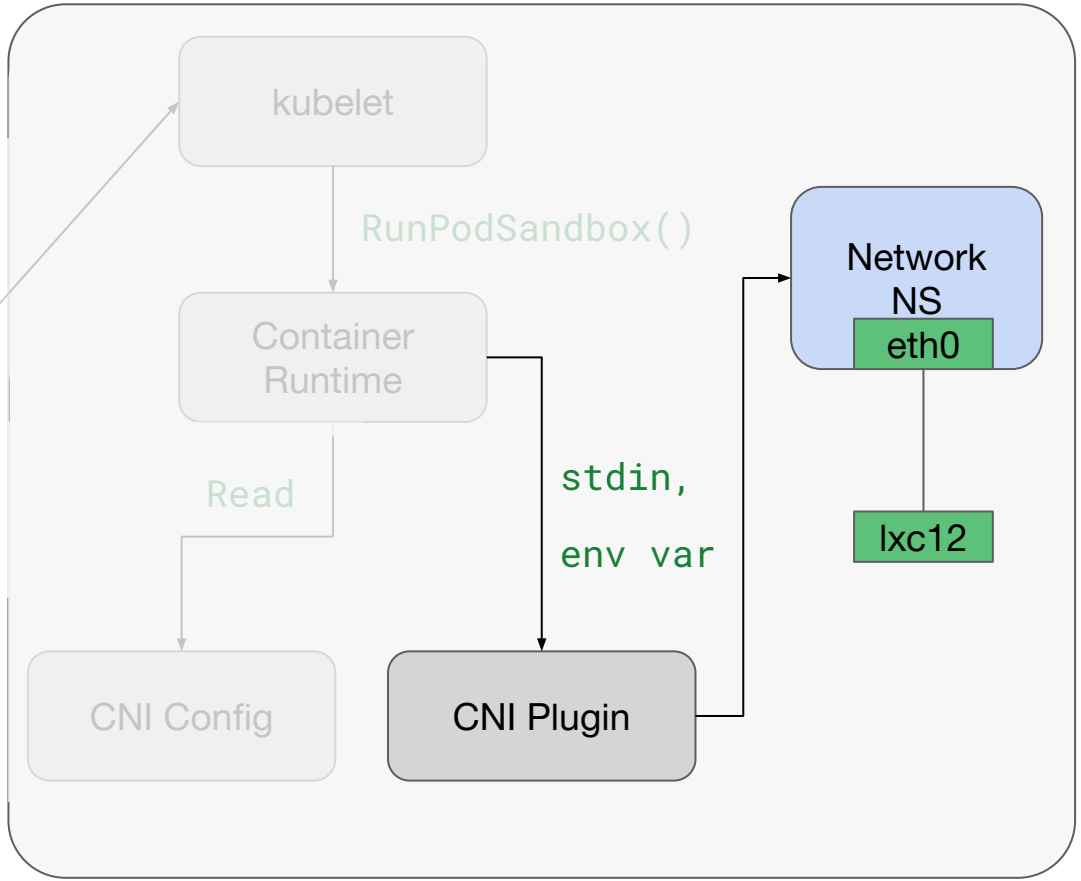
先備知識 - CNI

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: busybox
      image: busybox
```

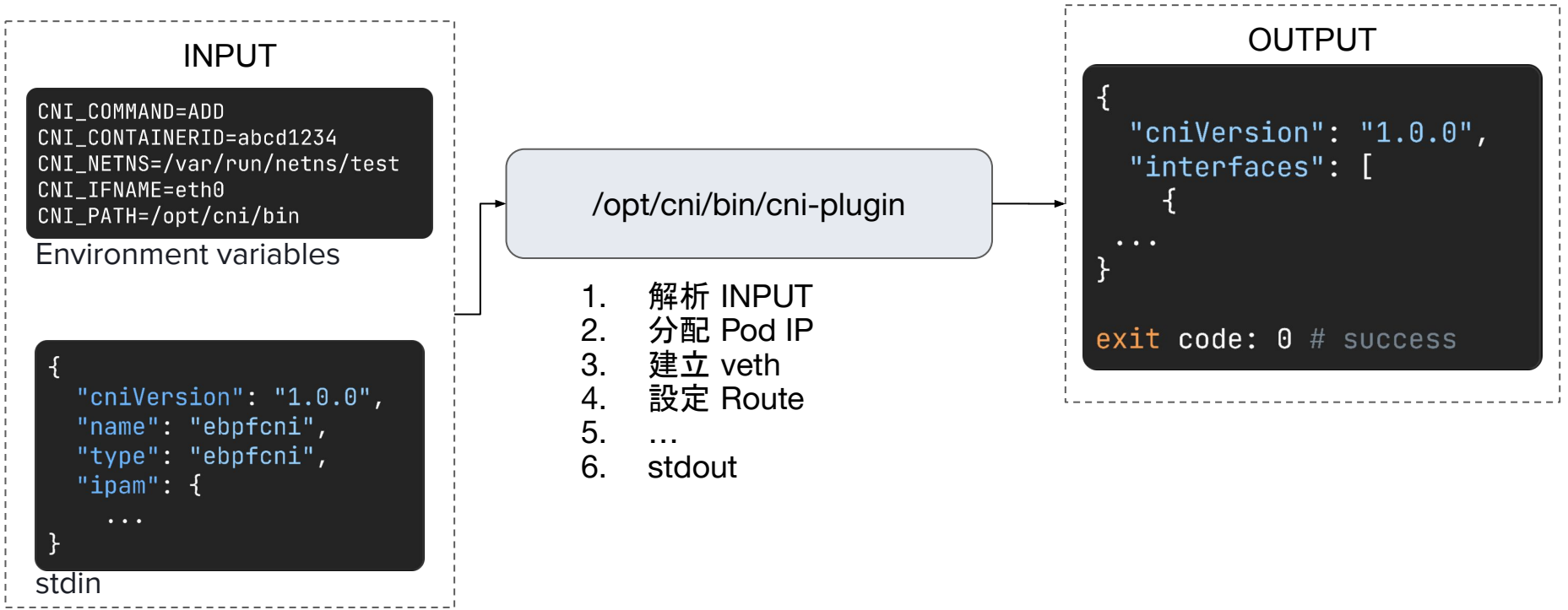


先備知識 - CNI

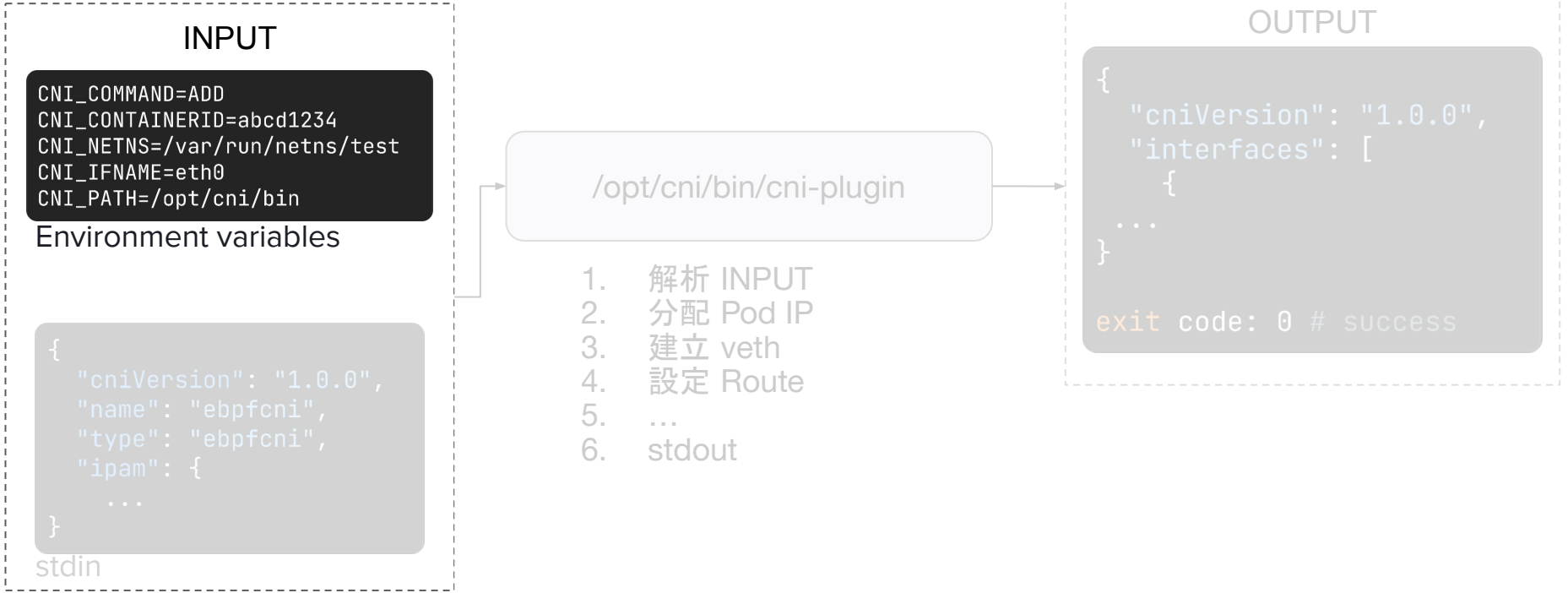
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: busybox
      image: busybox
```



先備知識 - CNI


INPUT
CNI_COMMAND=ADD
CNI_CONTAINERID=abcd1234
CNI_NETNS=/var/run/netns/test
CNI_IFNAME=eth0
CNI_PATH=/opt/cni/bin
Environment variables
{
"cniVersion": "1.0.0",
"name": "ebpfcni",
"type": "ebpfcni",
"ipam": {
...
}
stdin
/opt/cni/bin/cni-plugin
1. 解析 INPUT
2. 分配 Pod IP
3. 建立 veth
4. 設定 Route
5. …
6. stdout
OUTPUT
{
"cniVersion": "1.0.0",
"interfaces": [
{
...
}
exit code: 0 # success

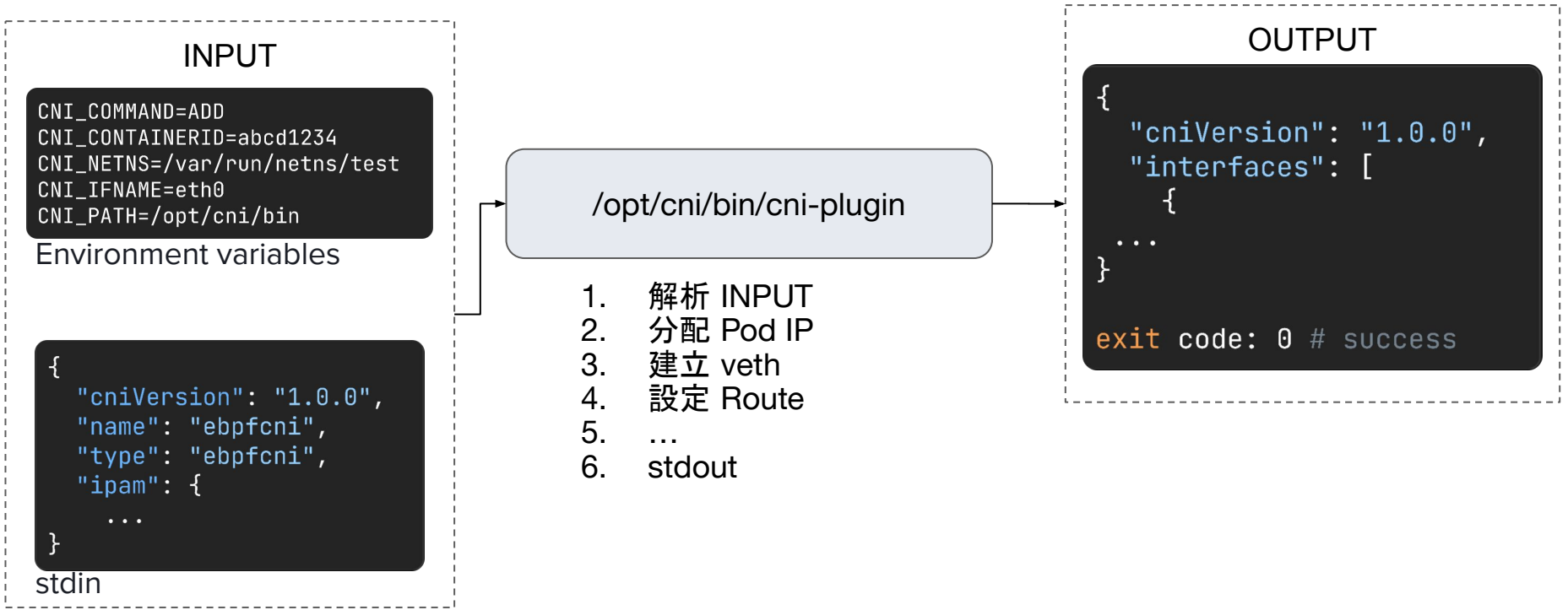
先備知識 - CNI


INPUT
CNI_COMMAND=ADD
CNI_CONTAINERID=abcd1234
CNI_NETNS=/var/run/netns/test
CNI_IFNAME=eth0
CNI_PATH=/opt/cni/bin
Environment variables
{
"cniVersion": "1.0.0",
"name": "ebpfcni",
"type": "ebpfcni",
"ipam": {
...
}
stdin
/opt/cni/bin/cni-plugin
1. 解析 INPUT
2. 分配 Pod IP
3. 建立 veth
4. 設定 Route
5. ...
6. stdout
OUTPUT
{
"cniVersion": "1.0.0",
"interfaces": [
{
...
}
exit code: 0 # success

環境變數	描述	範例
CNI_COMMAND	要執行的操作	ADD
CNI_CONTAINERID	容器識別碼	abcd1234
CNI_NETNS	網路命名空間的路徑	/var/run/netns/test
CNI_IFNAME	在容器內要建立的介面名稱	eth0
CNI_PATH	CNI 插件 binary 檔案的搜尋路徑	/opt/cni/bin
CNI_ARGS	由執行階段傳遞的額外參數	K8S_POD_NAME=...

https://github.com/containernetworking/cni/blob/v1.0.0/SPEC.md#parameters

先備知識 - CNI


INPUT
CNI_COMMAND=ADD
CNI_CONTAINERID=abcd1234
CNI_NETNS=/var/run/netns/test
CNI_IFNAME=eth0
CNI_PATH=/opt/cni/bin
Environment variables
{
"cniVersion": "1.0.0",
"name": "ebpfcni",
"type": "ebpfcni",
"ipam": {
...
}
stdin
/opt/cni/bin/cni-plugin
1. 解析 INPUT
2. 分配 Pod IP
3. 建立 veth
4. 設定 Route
5. …
6. stdout
OUTPUT
{
"cniVersion": "1.0.0",
"interfaces": [
{
...
}
exit code: 0 # success

第一章

動手寫基礎 CNI Plugin

目前環境

```
$ kubectl run my-pod
```

- K8s 1.32
- Single Node Cluster
- 尚未安裝 CNI Plugin（我們自己寫）

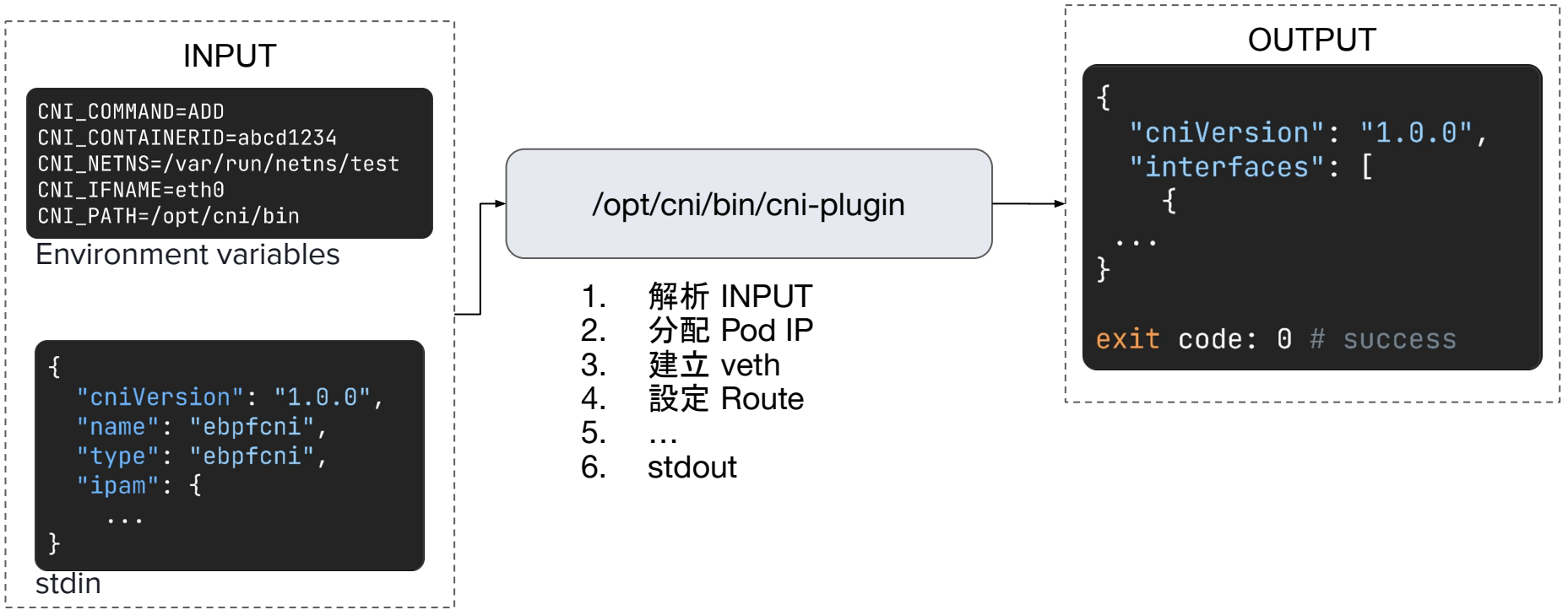
Node (NotReady)

Pending...

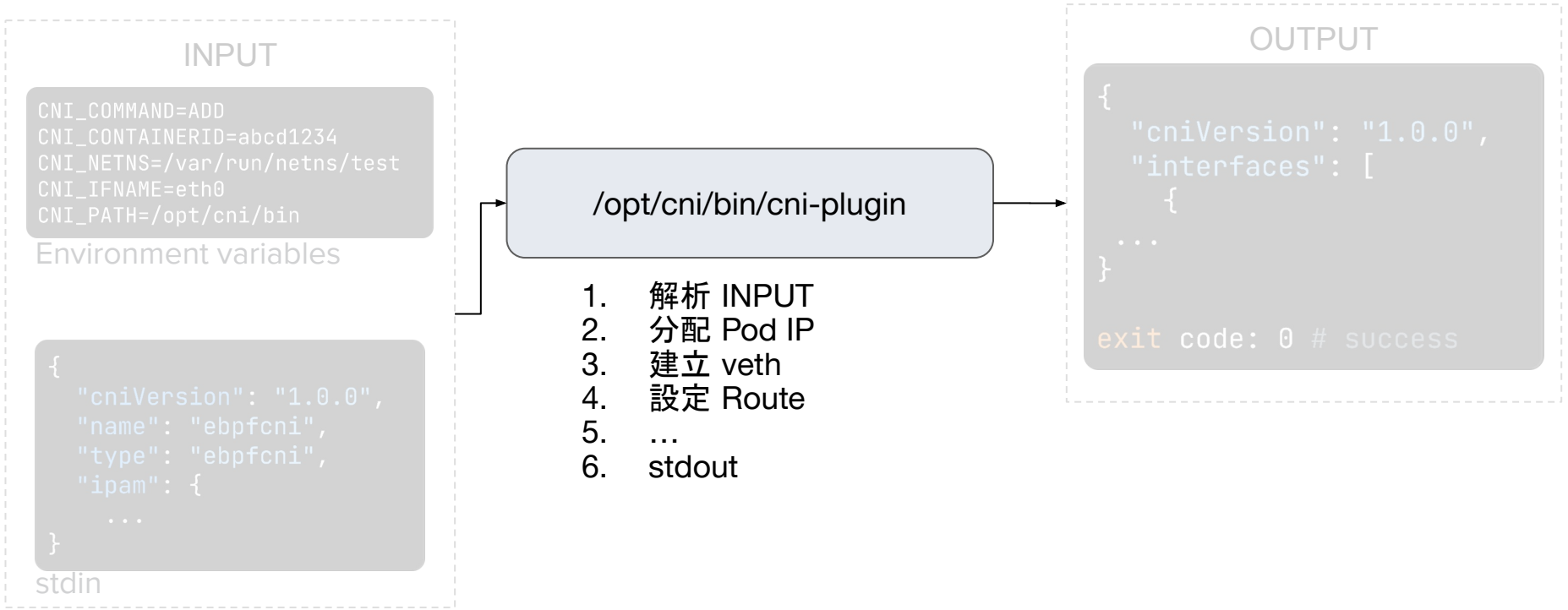
這是此時此刻的環境，還記得嗎？

Pod 卡在Pending

動手寫 CNI Plugin



動手寫 CNI Plugin



動手寫 CNI Plugin - ADD

1. 解析 INPUT (stdin, env var)
2. 分配 Pod IP
3. 建立 veth

```
# 讀取 Pod CIDR（從 stdin 傳進來的 JSON）
podcidr=$(cat /dev/stdin | jq -r ".podcidr")

# 從 Pod CIDR 計算 gateway IP，例如 10.13.0.1
gw_ip=$(echo $podcidr | sed "s:0/24:1:g")
```

```
# 隨機挑一個 IP（2-255）
n=$(( $RANDOM % 255 + 2 ))
ip=$(echo $podcidr | sed "s:0/24:$n:g")
```

```
# 建立 bridge（如果還沒存在）
brctl addbr bridge0
ip link set bridge0 up
ip addr add "${gw_ip}/24" dev bridge0

# 建立 veth pair: Pod-side ($CNI_IFNAME, 通常是 eth0) & Host-side ($host_ifname)
host_ifname="veth$n"
ip link add $CNI_IFNAME type veth peer name $host_ifname
ip link set $host_ifname up

# 把 host-side veth 接到 bridge
ip link set $host_ifname master bridge0
```

動手寫 CNI Plugin - ADD

1. 解析 INPUT (stdin, env var)
2. 分配 Pod IP
3. 建立 veth
4. 放進 Network NS
5. 設定 Route

```
# 建立 netns symlink
mkdir -p /var/run/netns/
ln -sfT $CNI_NETNS /var/run/netns/$CNI_CONTAINERID

# 移動 Pod-side veth (eth0) 進入 Pod netns
ip link set $CNI_IFNAME netns $CNI_CONTAINERID

# 隨機生成 MAC 並套用到 Pod-side veth
mac=$(printf '02:%02x:%02x:%02x:%02x:%02x' $[RANDOM%256]
$[RANDOM%256] $[RANDOM%256] $[RANDOM%256] $[RANDOM%256])
ip netns exec $CNI_CONTAINERID ip link set dev
$CNI_IFNAME address $mac
```

```
# 啟用 Pod-side veth
ip netns exec $CNI_CONTAINERID ip link set $CNI_IFNAME up

# 指定 Pod 的 IP (在 Pod-side veth 上)
ip netns exec $CNI_CONTAINERID ip addr add $ip/24 dev
$CNI_IFNAME

# 加入 default route 指向 gateway
ip netns exec $CNI_CONTAINERID ip route add default via $gw_ip
```

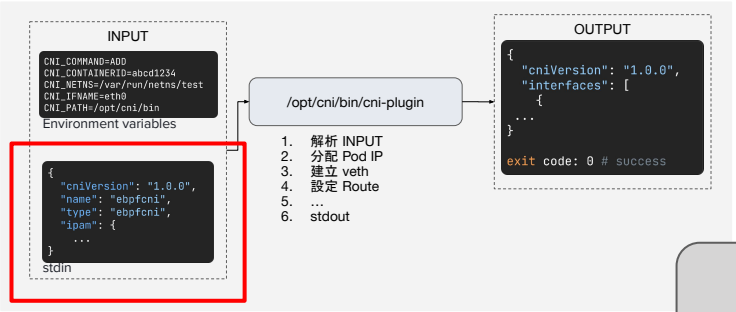
動手寫 CNI Plugin - ADD

1. 解析 INPUT (stdin, env var)
2. 分配 Pod IP
3. 建立 veth
4. 放進 Network NS
5. 設定 Route
6. 產出 OUTPUT (stdout)

```
address="${ip}/24"

output_template='
{
  "cniVersion": "1.0.0",
  "interfaces": [
      {
          "name": "%s",
          "mac": "%s",
          "sandbox": "%s"
      }
  ],
  "ips": [
      {
          "version": "4",
          "address": "%s",
          "gateway": "%s",
          "interface": 0
      }
  ]
}'

printf "${output_template}" $CNI_IFNAME $mac
$CNI_NETNS $address $gw_ip
```

動手寫 CNI Plugin - ADD

1. 解析 INPUT (stdin, env var)
2. 分配 Pod IP
3. 建立 veth
4. 放進 Network NS
5. 設定 Route
6. 產出 OUTPUT (stdout)
7. 創建 CNI Config



CNI Plugin INPUT & Output

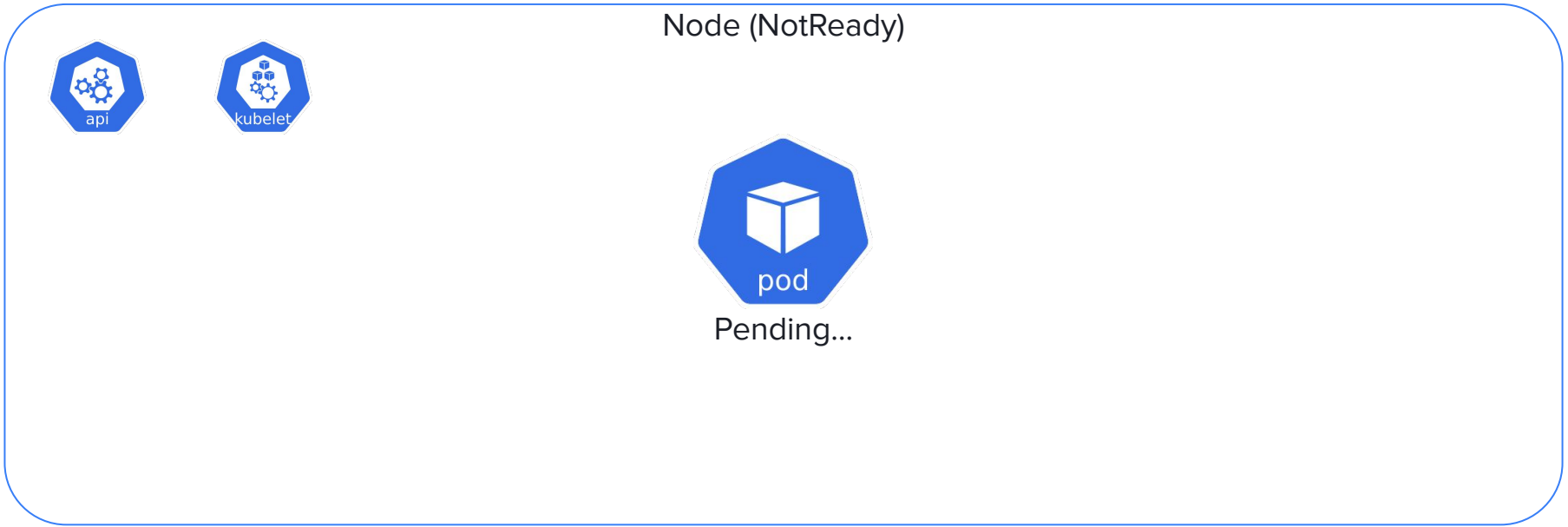
記得創建 CNI Config

```
sudo tee /etc/cni/net.d/10-ebpfcni.conf > /dev/null <<'EOF'
{
    "cniVersion":"1.0.0",
    "name":"ebpfcni",
    "type":"ebpfcni",
    "podcidr":"10.13.0.0/24"
}
EOF
```

目前環境

```
$ kubectl run my-pod
```

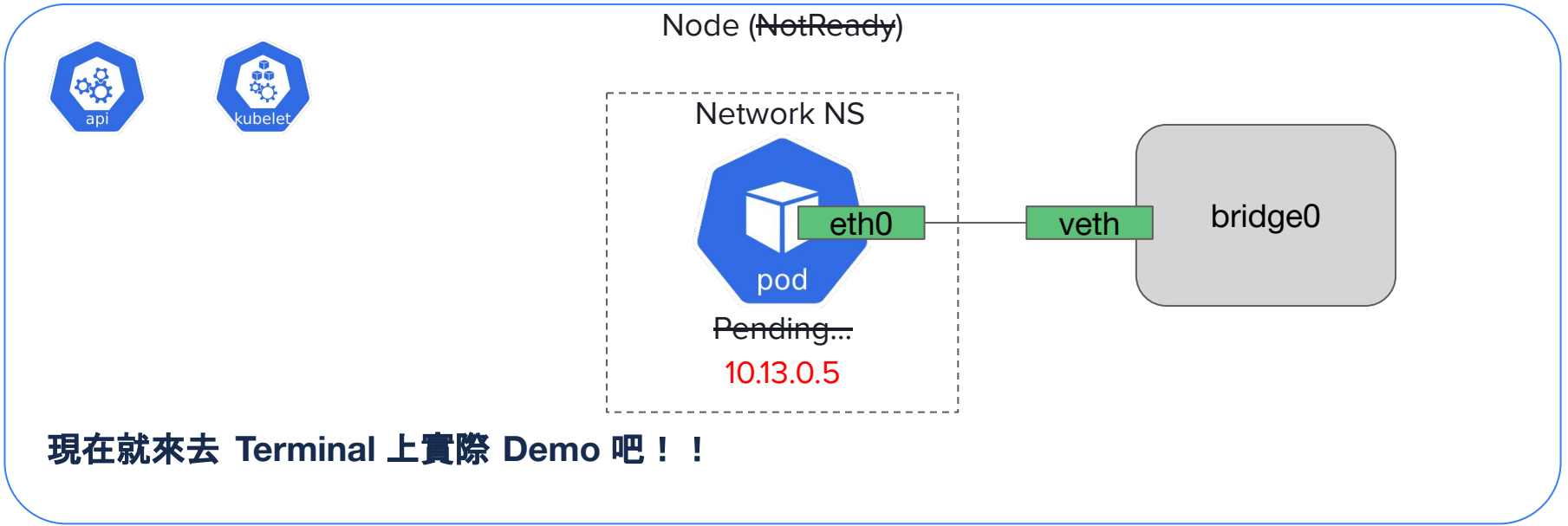
- K8s 1.32
- Single Node Cluster
- 尚未安裝 CNI Plugin（我們自己寫）



預期環境

```
$ kubectl run my-pod
```

✅ 寫好的 CNI Plugin 已放到 /opt/cni/bin/
✅ 已創建好 /etc/cni/net.d/10-ebpfcni.conf

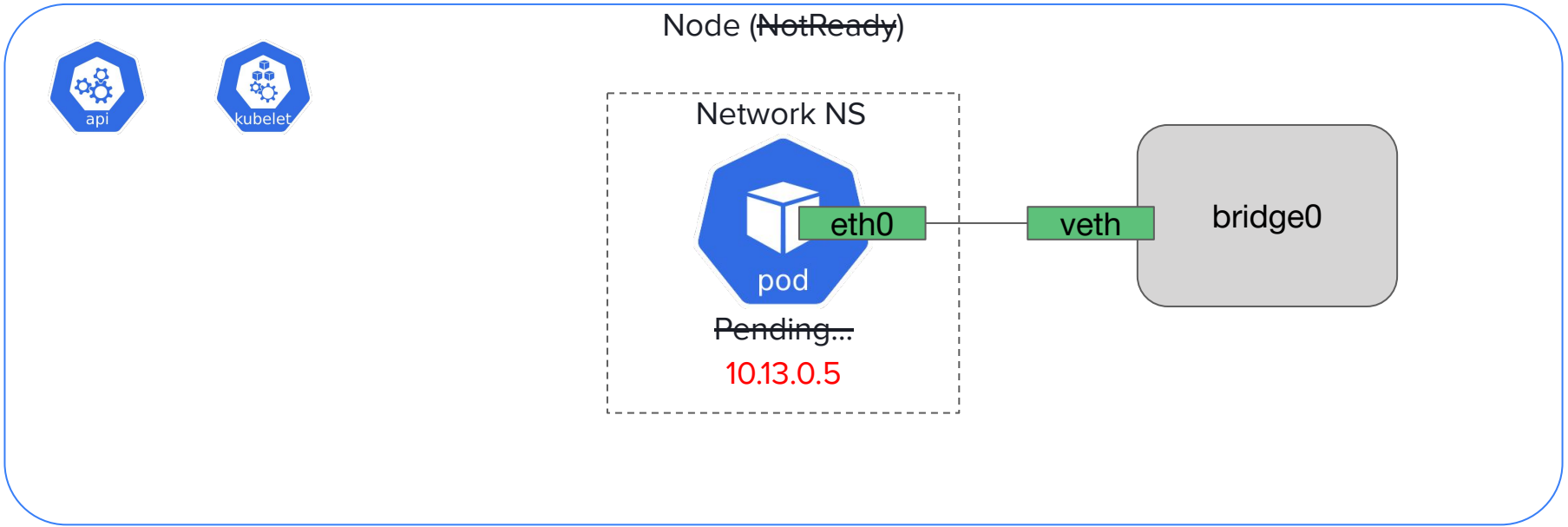


現在就來去 Terminal 上實際 Demo 吧！！

預期環境

```
$ kubectl run my-pod
```

✅ 寫好的 CNI Plugin 已放到 /opt/cni/bin/
✅ 已創建好 /etc/cni/net.d/10-ebpfcni.conf



第二章

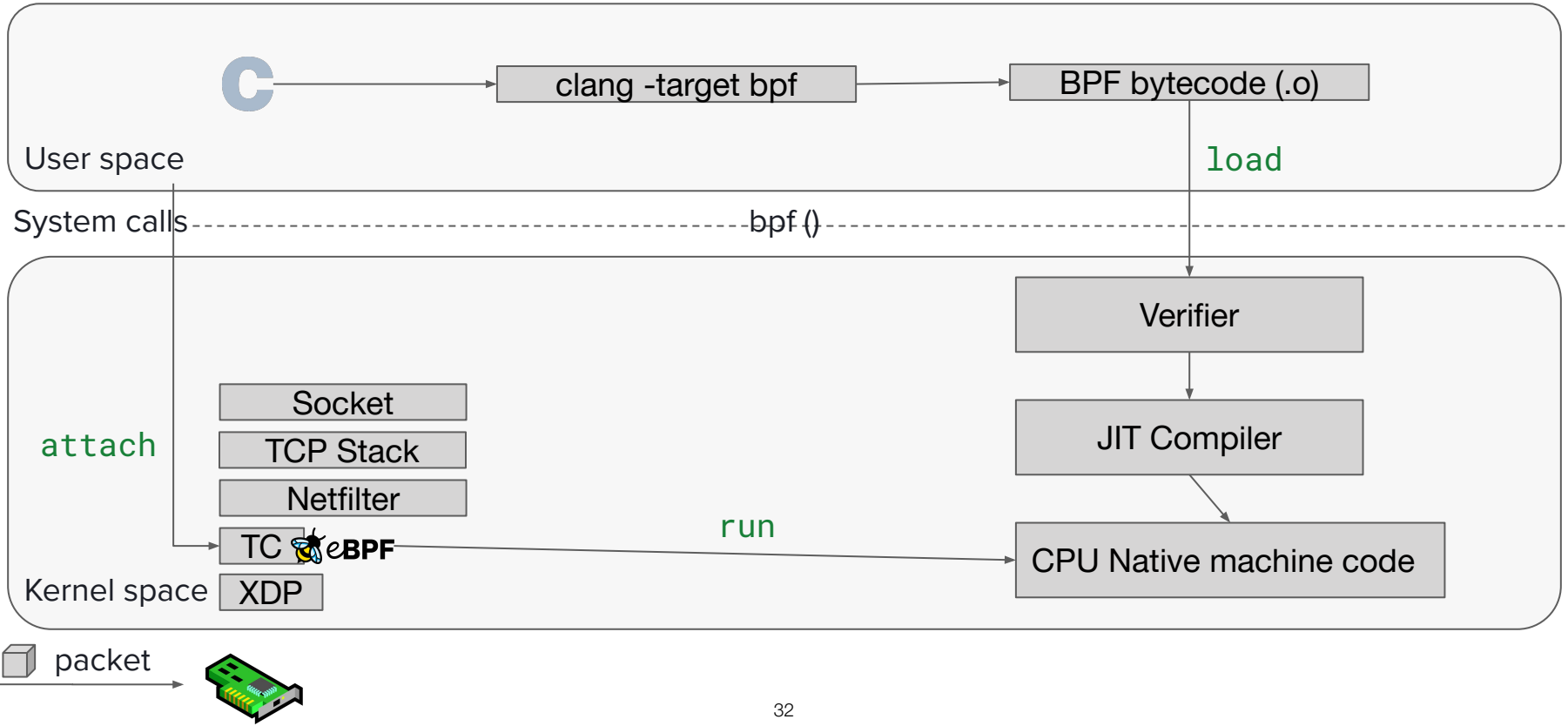
利用 eBPF 赋予 Kernel 可程式化的能力

eBPF 是什麼？

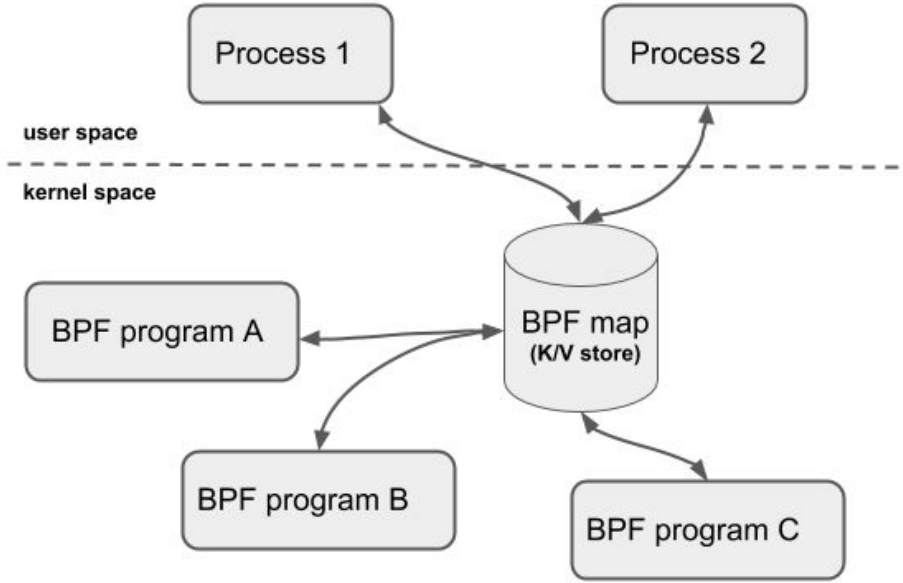
一種不用修改 Linux Kernel 程式碼或是重新編譯 Kernel

可以直接在 Linux Kernel 中動態插入程式碼的技術

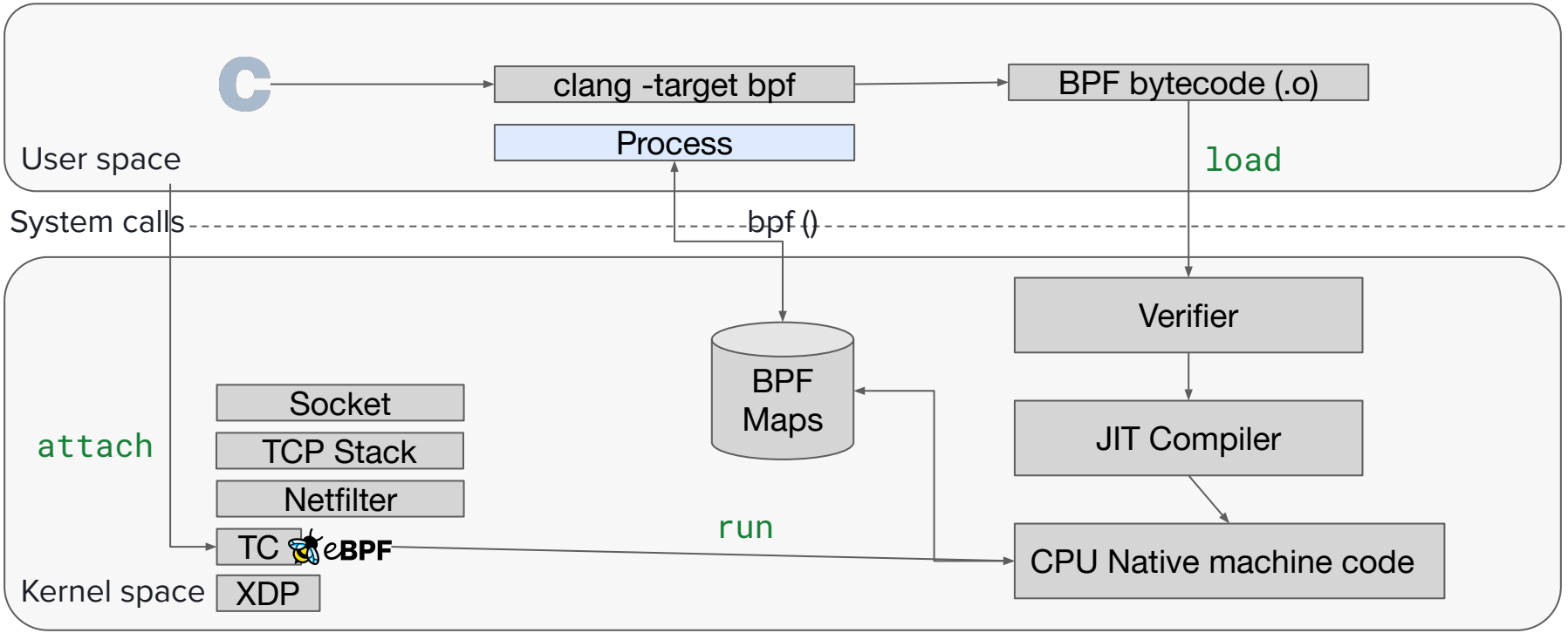
eBPF 是什麼？


C
clang -target bpf
BPF bytecode (.o)
User space
load
System calls
bpf ()
Verifier
Socket
TCP Stack
Netfilter
attach
JIT Compiler
TC
eBPF
run
CPU Native machine code
Kernel space
XDP
packet

BPF Map


Process 1
Process 2
user space
kernel space
BPF map
(K/V store)
BPF program A
BPF program B
BPF program C

eBPF 是什麼？

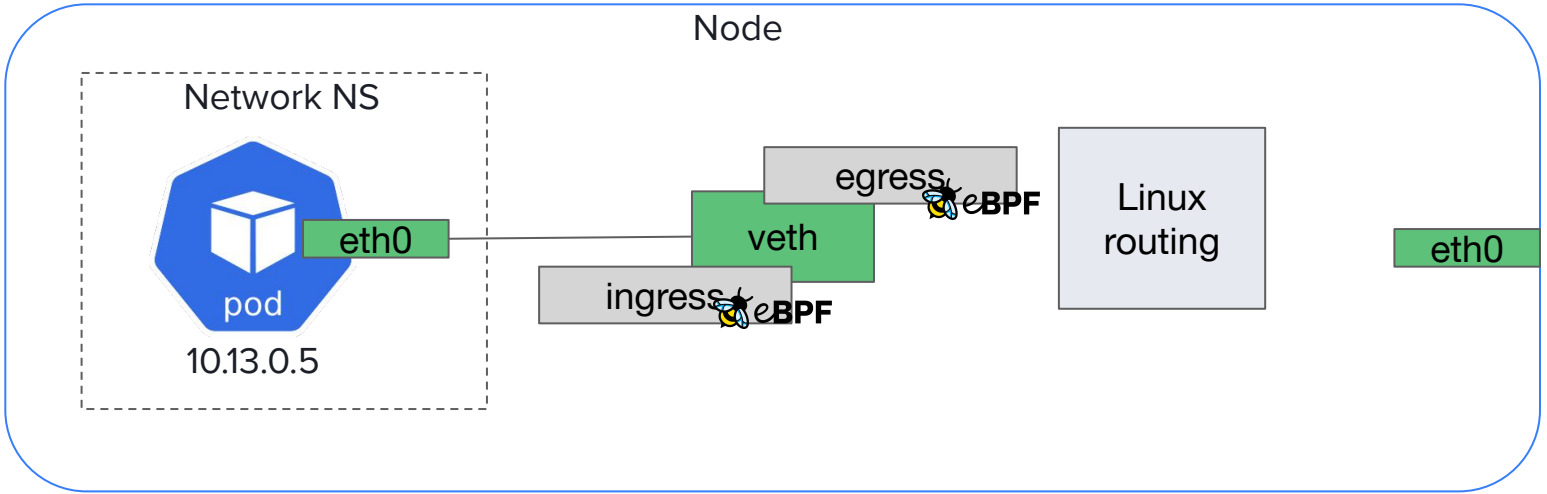

clang -target bpf
BPF bytecode (.o)
Process
User space
load
System calls
bpf ()
Verifier
BPF
Maps
Socket
TCP Stack
Netfilter
attach
JIT Compiler
run
TC
eBPF
CPU Native machine code
Kernel space
XDP

先備知識 - Traffic Control (TC)

- 一個複雜的子系統和框架... 可以用來做流量整形
- 在 net device 身上, Kernel 提供 ingress 和 egress hook point
- 這兩個 hook point 由 TC 管理 / 暴露 API 供我們掛載 eBPF Prog 到 hook point
- 推薦閱讀論文: Linux Traffic Control Classifier-Action Subsystem Architecture - Jamal Hadi Salim

先備知識 - Traffic Control (TC)

- 一個複雜的子系統和框架... 可以用來做流量整形
- 在 net device 身上, Kernel 提供 ingress 和 egress hook point
- 這兩個 hook point 由 TC 管理 / 暴露 API 供我們掛載 eBPF Prog 到 hook point
- 推薦閱讀論文: Linux Traffic Control Classifier-Action Subsystem Architecture - Jamal Hadi Salim



SHOPLINE

TC, Hello World eBPF Prog

```c
#include "vmlinux.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>

#define TC_ACT_OK 0
#define TC_ACT_SHOT 2

char _license[] SEC("license") = "GPL";

// 定義 TC 程式的 section
SEC("tc")
int simple_tc_parser(struct __sk_buff *skb)
{
    // 取得封包資料的起始與結束位置
    void *data_end = (void *)(long)skb->data_end;
    void *data = (void *)(long)skb->data;

    struct ethhdr *eth = data;
    struct iphdr *iph = data + sizeof(*eth);

    // 邊界檢查，避免讀超過封包長度
    if ((void *)iph + sizeof(*iph) > data_end)
        return TC_ACT_SHOT;

    // 印出封包資訊
    bpf_printk("TC: Got a packet from 0x%x to 0x%x",
               bpf_ntohl(iph->saddr), bpf_ntohl(iph->daddr));

    // 放行封包
    return TC_ACT_OK;
}
```

TC, Hello World eBPF Prog

```c
#include "vmlinux.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>

#define TC_ACT_OK 0
#define TC_ACT_SHOT 2

char _license[] SEC("license") = "GPL";

// 定義 TC 程式的 section
SEC("tc")
int simple_tc_parser(struct __sk_buff *skb)
{
    // 取得封包資料的起始與結束位置
    void *data_end = (void *)(long)skb->data_end;
    void *data = (void *)(long)skb->data;

    struct ethhdr *eth = data;
    struct iphdr *iph = data + sizeof(*eth);

    // 邊界檢查，避免讀超過封包長度
    if ((void *)iph + sizeof(*iph) > data_end)
        return TC_ACT_SHOT;

    // 印出封包資訊
    bpf_printk("TC: Got a packet from 0x%x to 0x%x",
               bpf_ntohl(iph->saddr), bpf_ntohl(iph->daddr));

    // 放行封包
    return TC_ACT_OK;
}
```

Return Code	Value	Meaning
TC_ACT_OK	0	允許通過 (pass)
TC_ACT_SHOT	2	丟棄封包 (drop)
TC_ACT_REDIRECT	7	導向其他介面 /queue (redirect)

Direct Action Mode - Return Code 表格整理

TC, Hello World eBPF Prog

```c
#include "vmlinux.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>

#define TC_ACT_OK 0
#define TC_ACT_SHOT 2

char _license[] SEC("license") = "GPL";

// 定義 TC 程式的 section
SEC("tc")
int simple_tc_parser(struct __sk_buff *skb)
{
    // 取得封包資料的起始與結束位置
    void *data_end = (void *)(long)skb->data_end;
    void *data = (void *)(long)skb->data;

    struct ethhdr *eth = data;
    struct iphdr *iph = data + sizeof(*eth);

    // 邊界檢查，避免讀超過封包長度
    if ((void *)iph + sizeof(*iph) > data_end)
        return TC_ACT_SHOT;

    // 印出封包資訊
    bpf_printk("TC: Got a packet from 0x%x to 0x%x",
               bpf_ntohl(iph->saddr), bpf_ntohl(iph->daddr));

    // 放行封包
    return TC_ACT_OK;
}
```

Return Code	Value	Meaning
TC_ACT_OK	0	允許通過 (pass)
TC_ACT_SHOT	2	丟棄封包 (drop)
TC_ACT_REDIRECT	7	導向其他介面 /queue (redirect)

Direct Action Mode - Return Code 表格整理

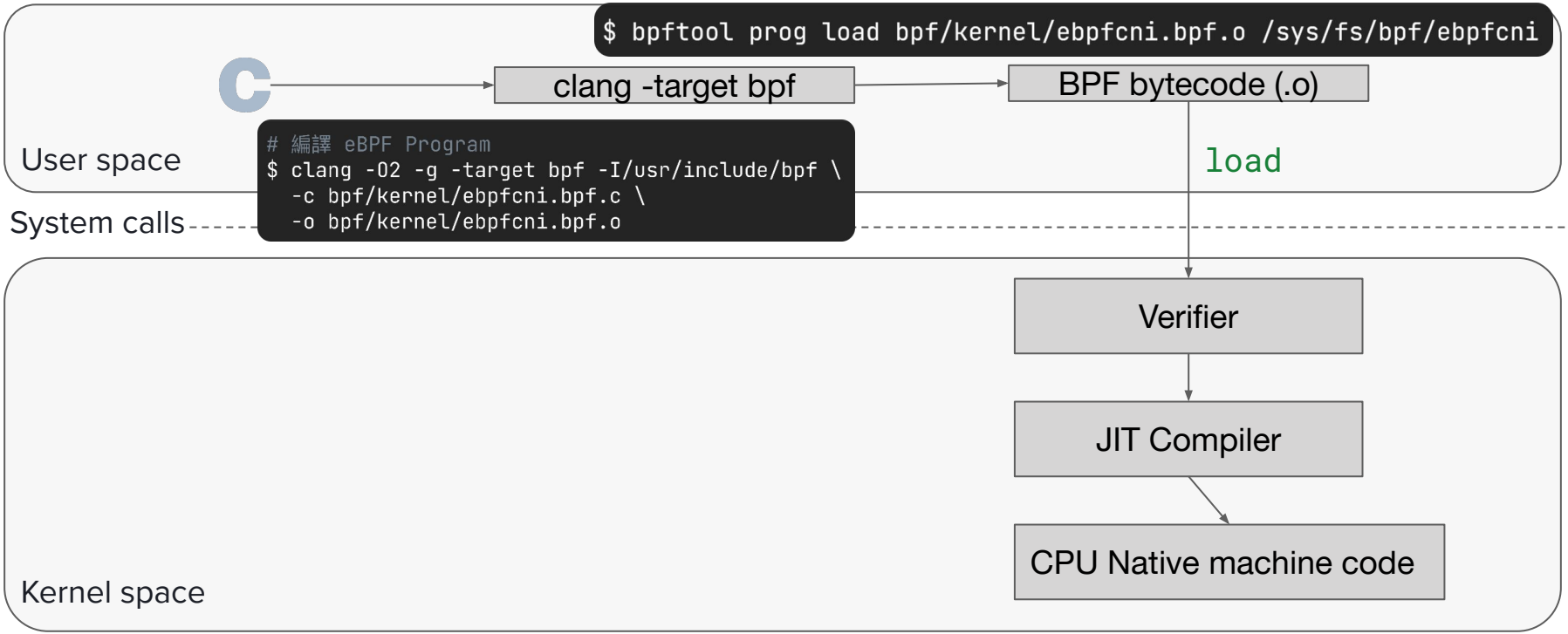
TC, Hello World eBPF Prog

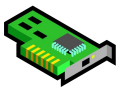
```c
#include "vmlinux.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>

#define TC_ACT_OK 0
#define TC_ACT_SHOT 2

char _license[] SEC("license") = "GPL";

// 定義 TC 程式的 section
SEC("tc")
int simple_tc_parser(struct __sk_buff *skb)
{
    // 取得封包資料的起始與結束位置
    void *data_end = (void *)(long)skb->data_end;
    void *data = (void *)(long)skb->data;

    struct ethhdr *eth = data;
    struct iphdr *iph = data + sizeof(*eth);

    // 邊界檢查，避免讀超過封包長度
    if ((void *)iph + sizeof(*iph) > data_end)
        return TC_ACT_SHOT;

    // 印出封包資訊
    bpf_printk("TC: Got a packet from 0x%x to 0x%x",
               bpf_ntohl(iph->saddr), bpf_ntohl(iph->daddr));

    // 放行封包
    return TC_ACT_OK;
}
```

Return Code	Value	Meaning
TC_ACT_OK	0	允許通過 (pass)
TC_ACT_SHOT	2	丟棄封包 (drop)
TC_ACT_REDIRECT	7	導向其他介面 /queue (redirect)

Direct Action Mode - Return Code 表格整理

eBPF 是什麼？


$ bpftool prog load bpf/kernel/ebpfcni.bpf.o /sys/fs/bpf/ebpfcni
C
clang -target bpf
BPF bytecode (.o)
User space
# 編譯 eBPF Program
$ clang -O2 -g -target bpf -I/usr/include/bpf \
-c bpf/kernel/ebpfcni.bpf.c \
-o bpf/kernel/ebpfcni.bpf.o
load
System calls
Verifier
JIT Compiler
CPU Native machine code
Kernel space

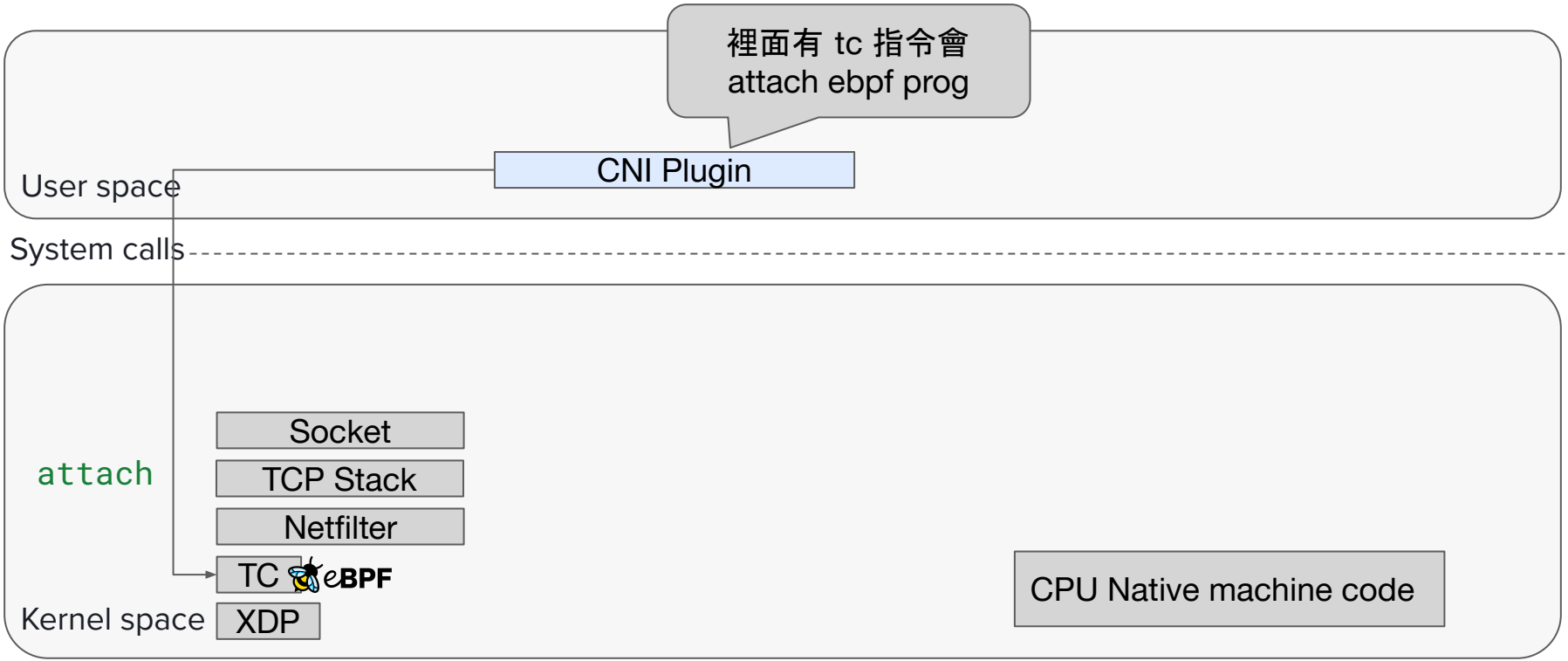


具體如何 attach?

待會這段指令要放到
cni-plugin 的程式碼

```
tc qdisc add dev $host_ifname clsact
tc filter add dev $host_ifname ingress bpf direct-action object-pinned /sys/fs/bpf/ebpfcni
```

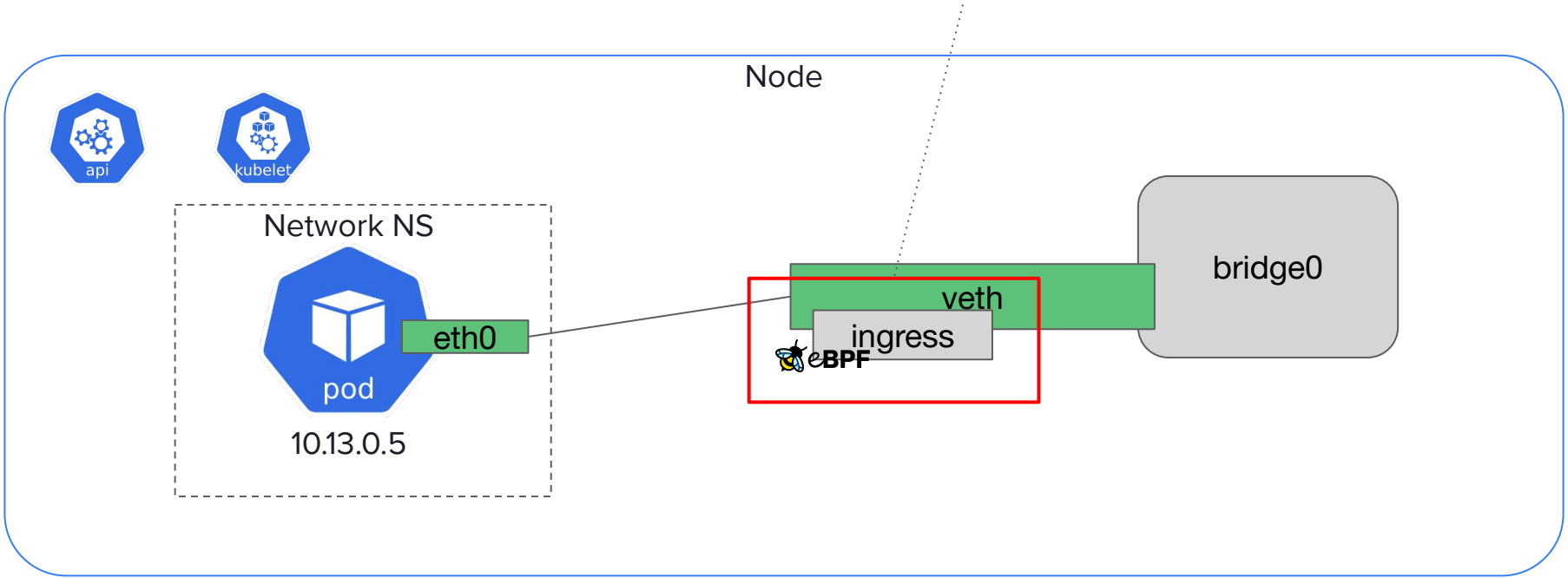
讓 CNI Plugin 來幫我們 attach



預期環境

```
$ kubectl run my-pod
```

- CNI Plugin 會幫我們在 host-side veth 的 TC Ingress attach eBPF Prog



來實際 Demo - 賦予 eBPF & Debugging

看 attach 什麼 eBPF Prog

```
$ tc filter show dev <dev> ingress
$ tc filter show dev <dev> egress
```

```
$ bpftool net
```

看 bpf_printk()

```
$ sudo cat /sys/kernel/debug/tracing/trace_pipe
```

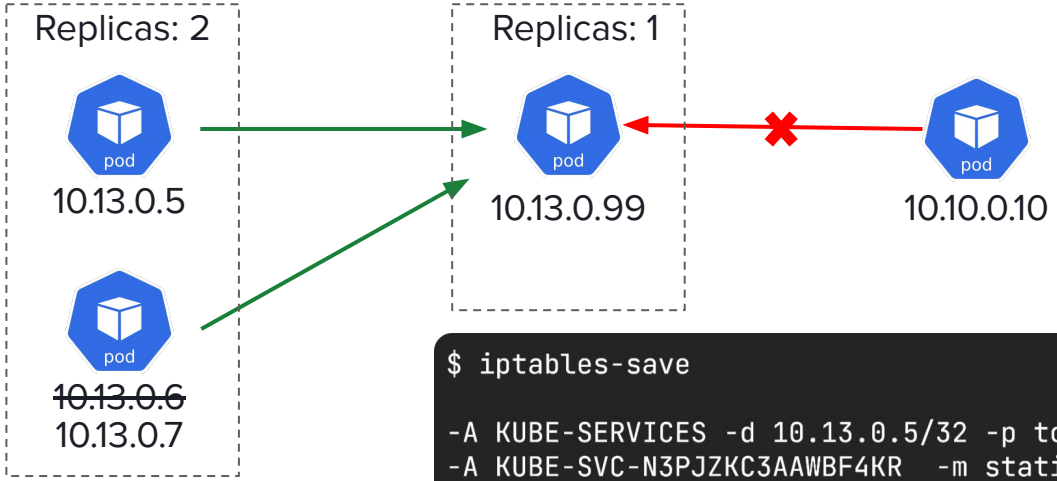
```
$ bpftool prog tracelog
```

第三章

NetworkPolicy /

在 eBPF 實現 Identity-based 封包決策

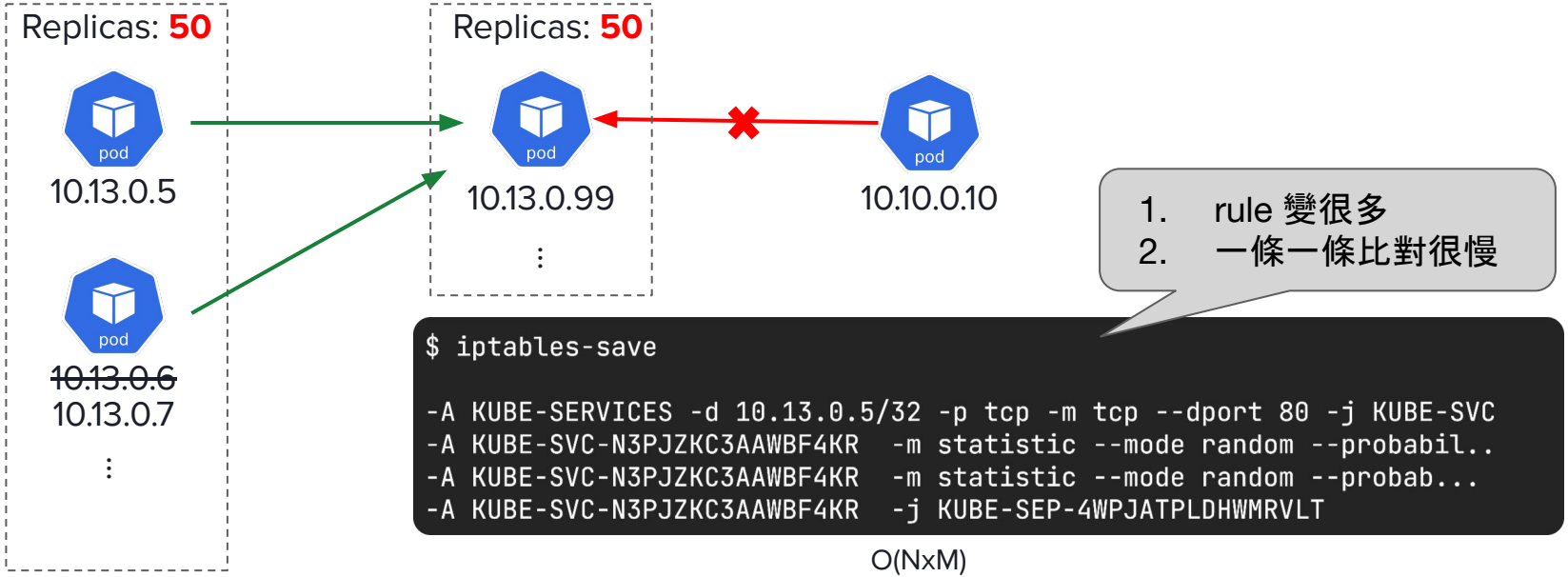
iptables 實現 Allow/Deny




```
$ iptables-save

-A KUBE-SERVICES -d 10.13.0.5/32 -p tcp -m tcp --dport 80 -j KUBE-SVC
-A KUBE-SVC-N3PJZKC3AAWBF4KR  -m statistic --mode random --probabil..
-A KUBE-SVC-N3PJZKC3AAWBF4KR  -m statistic --mode random --probab...
-A KUBE-SVC-N3PJZKC3AAWBF4KR  -j KUBE-SEP-4WPJATPLDHWMRVLT
```

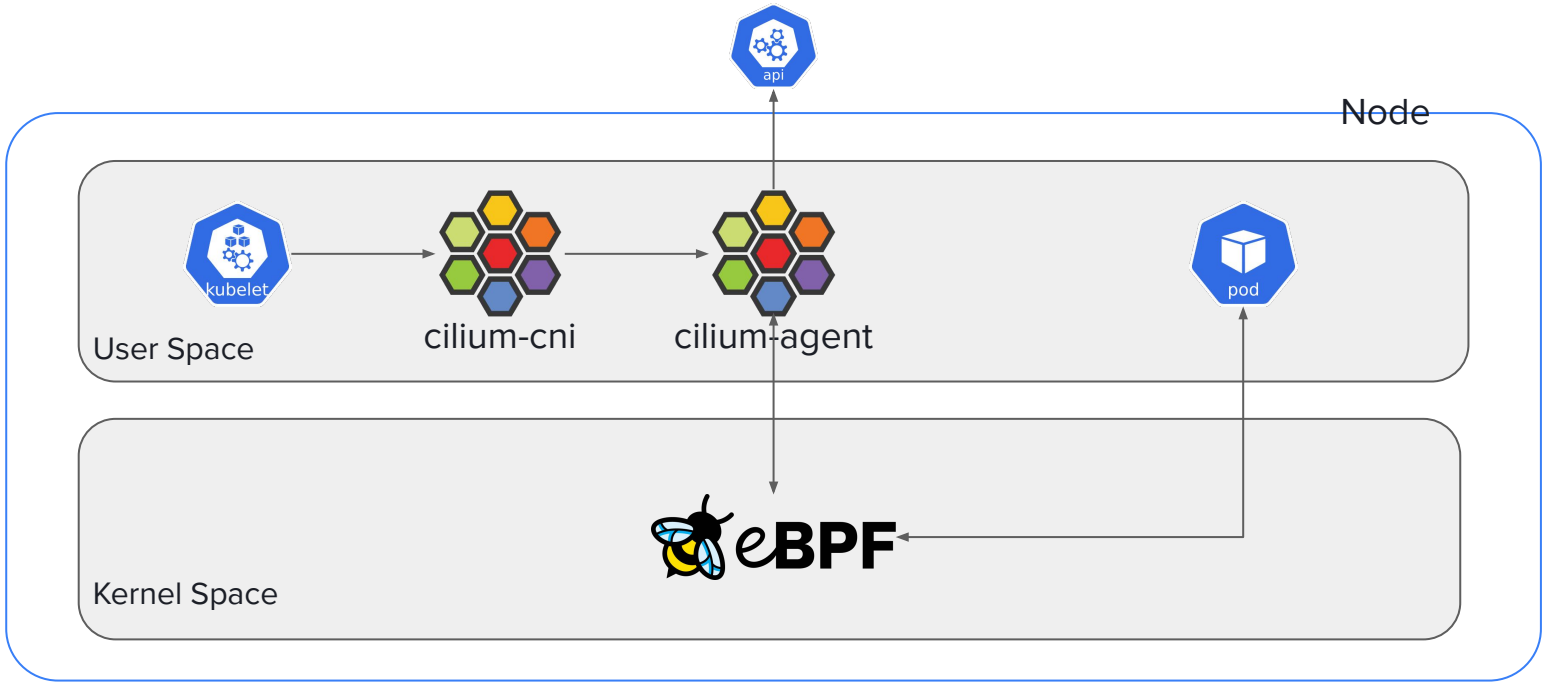
O(NxM)
N: 來源 Pod 數量
M: 目的 Pod 數量

iptables 實現 Allow/Deny

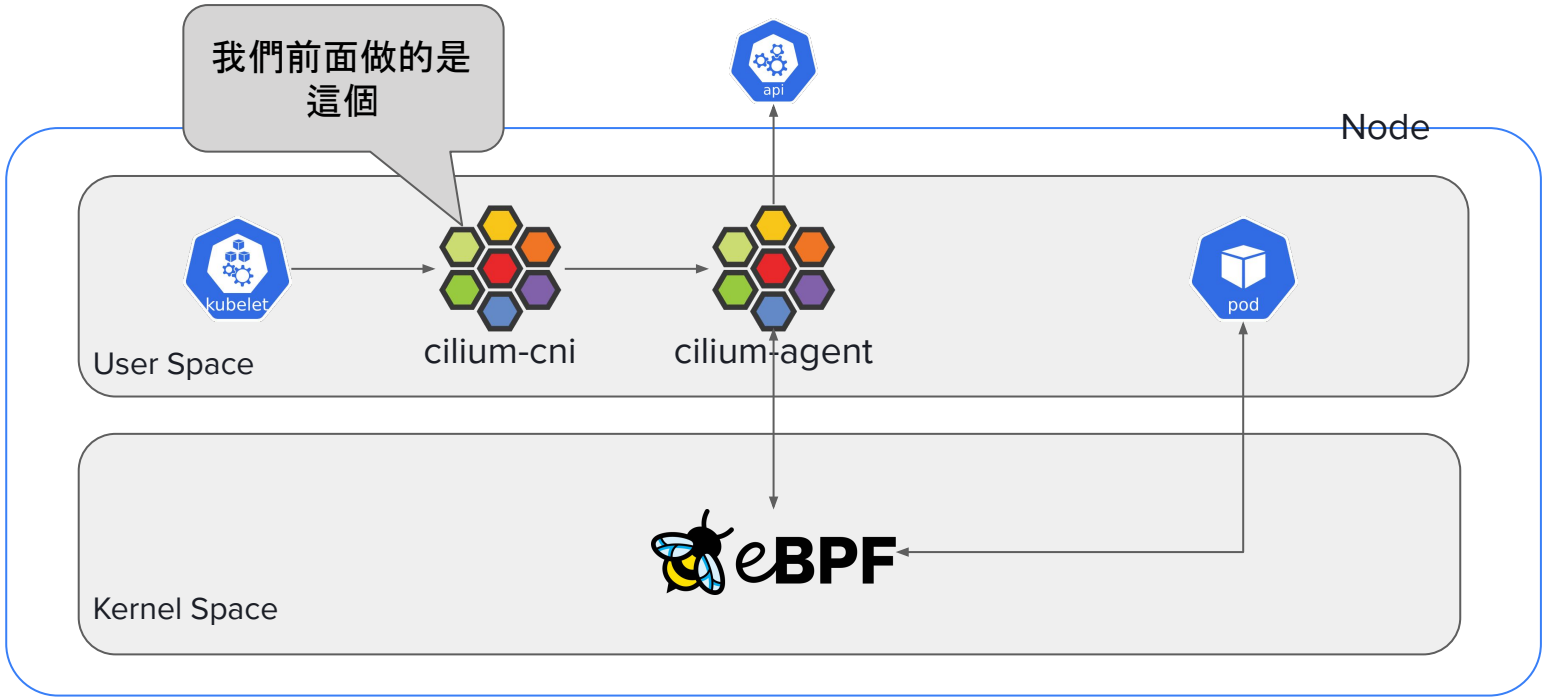


O(NxM)
N: 來源 Pod 數量
M: 目的 Pod 數量

Cilium 基本架構

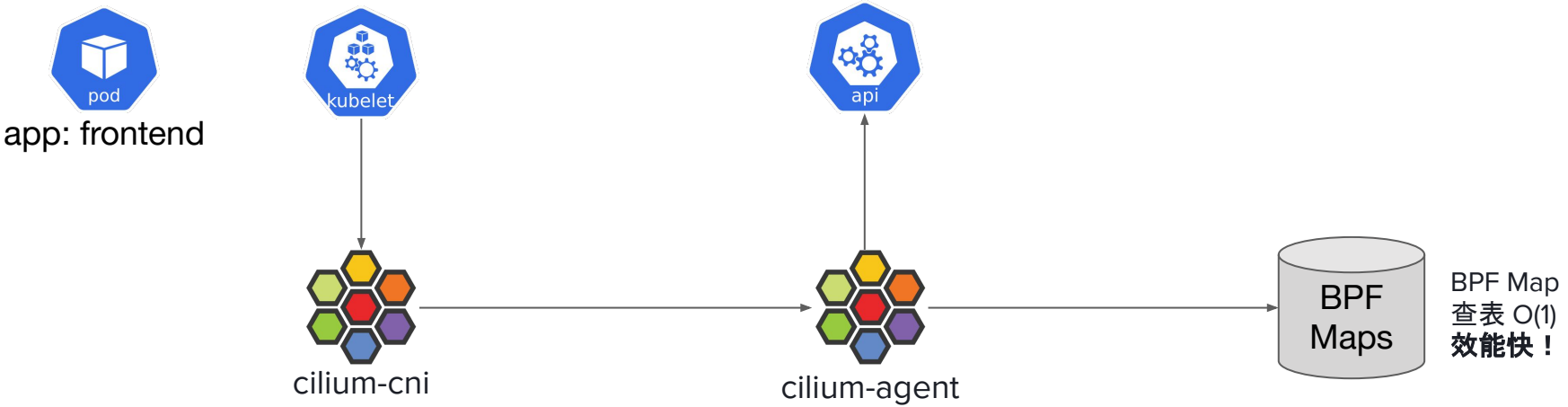

api
Node
kubelet
cilium-cni
cilium-agent
pod
User Space
eBPF
Kernel Space

Cilium 基本架構

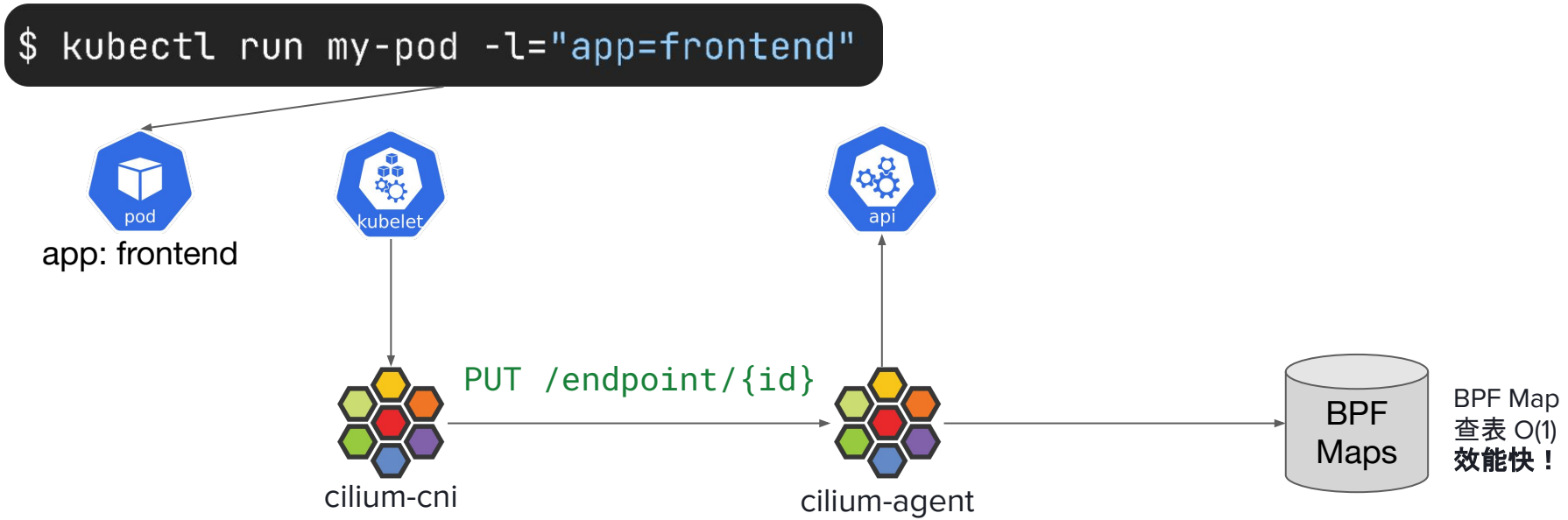

我們前面做的是
這個
api
Node
kubelet
cilium-cni
cilium-agent
pod
User Space
eBPF
Kernel Space

先備知識 - Cilium 如何實現 NetworkPolicy

```
$ kubectl run my-pod -l="app=frontend"
```

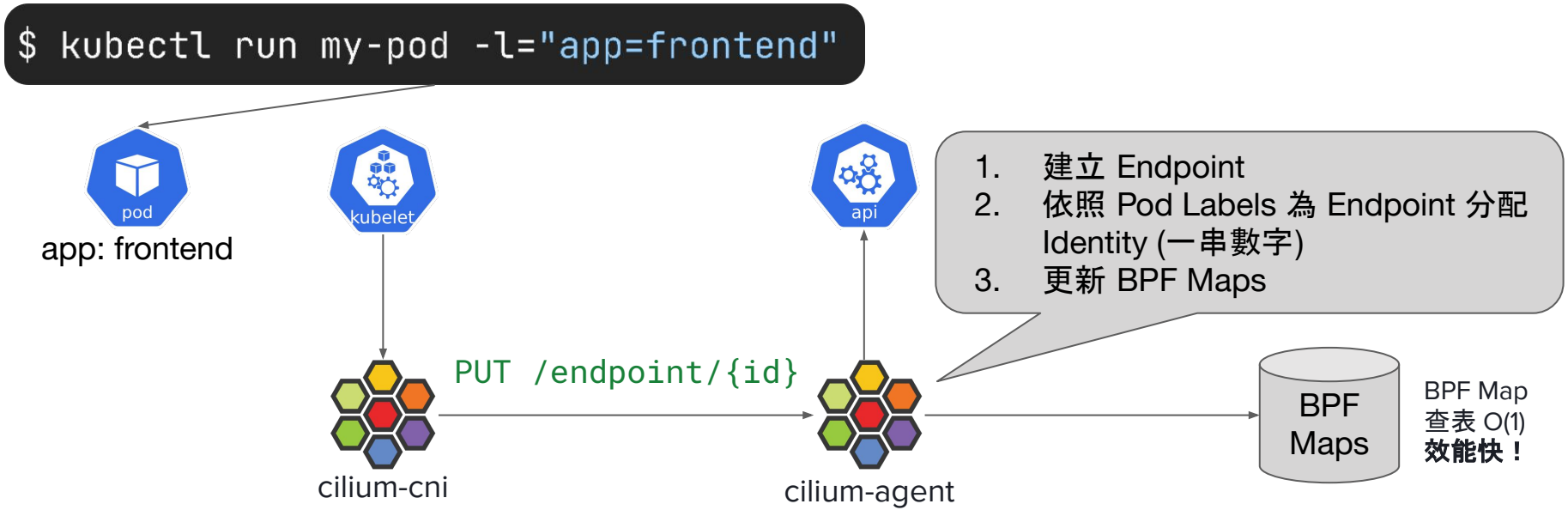


先備知識 - Cilium 如何實現 NetworkPolicy



SHOPLINE

先備知識 - Cilium 如何實現 NetworkPolicy



SHOPLINE

先備知識 - Cilium 如何實現 NetworkPolicy

Pod	Lables	Identity
my-pod-1	app: frontend	
my-pod-2	app: frontend	
my-pod-3	app: frontend, env: staging	

猜猜看 my-pod-3 的 Identity 會跟 my-pod-1 或 my-pod-2 一模一樣嗎？

SHOPLINE

先備知識 - Cilium 如何實現 NetworkPolicy

Pod	Lables	Identity
my-pod-1	app: frontend	
my-pod-2	app: frontend	
my-pod-3	app: frontend, env: staging	

```
❯ kubectl get ciliumendpoint
NAME       SECURITY IDENTITY   ENDPOINT STATE   IPV4           IPV6
my-pod-1   156897              ready            [redacted].44
my-pod-2   156897              ready            [redacted].49
my-pod-3   142483              ready            [redacted].1
```

先備知識 - Cilium 如何實現 NetworkPolicy

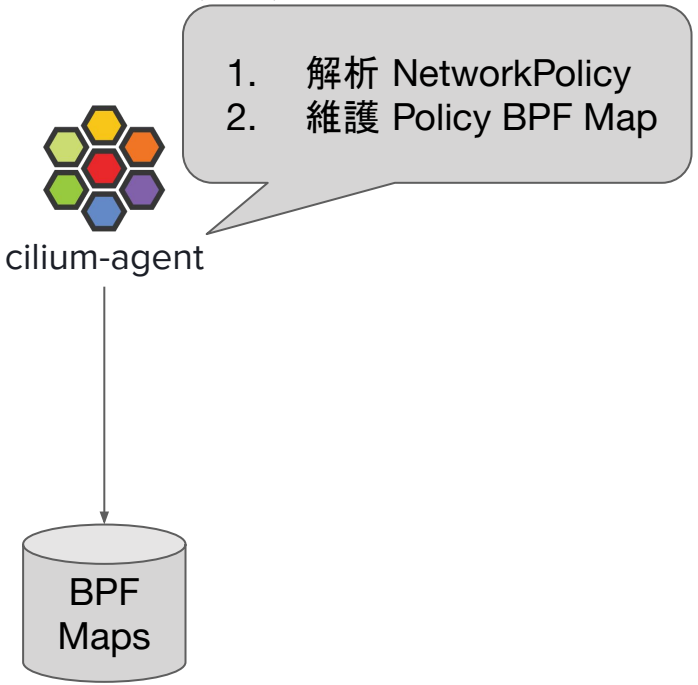
Pod	Lables	Identity
my-pod-1	app: frontend	156897
my-pod-2	app: frontend	156897
my-pod-3	app: frontend, env: staging	142483

```
❯ kubectl get ciliumendpoint
NAME       SECURITY IDENTITY   ENDPOINT STATE   IPV4           IPV6
my-pod-1   156897              ready            [redacted].44
my-pod-2   156897              ready            [redacted].49
my-pod-3   142483              ready            [redacted].1
```

先備知識 - Cilium 如何實現 NetworkPolicy

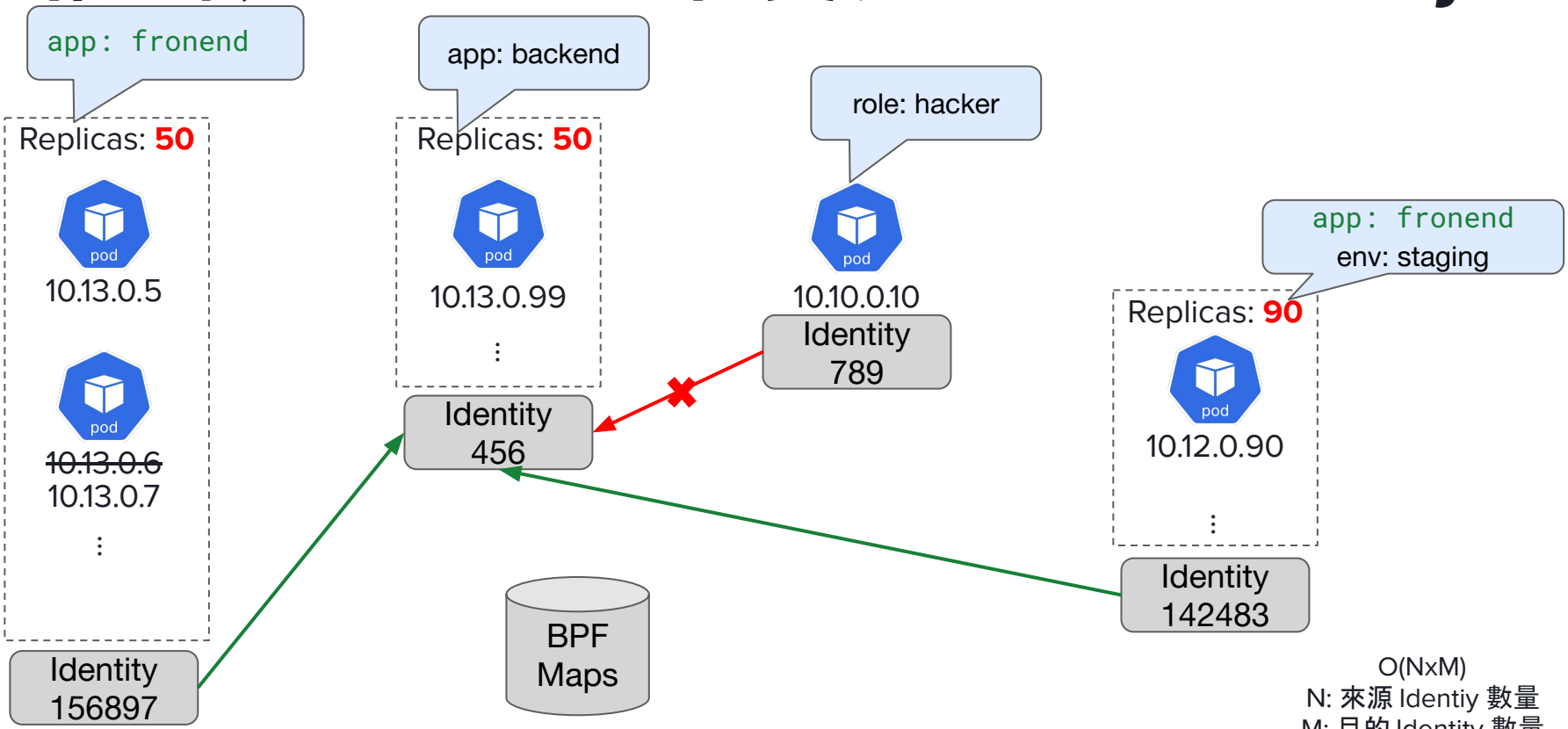
僅允許 app: fronend 的 Ingress 流量

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: frontend
```



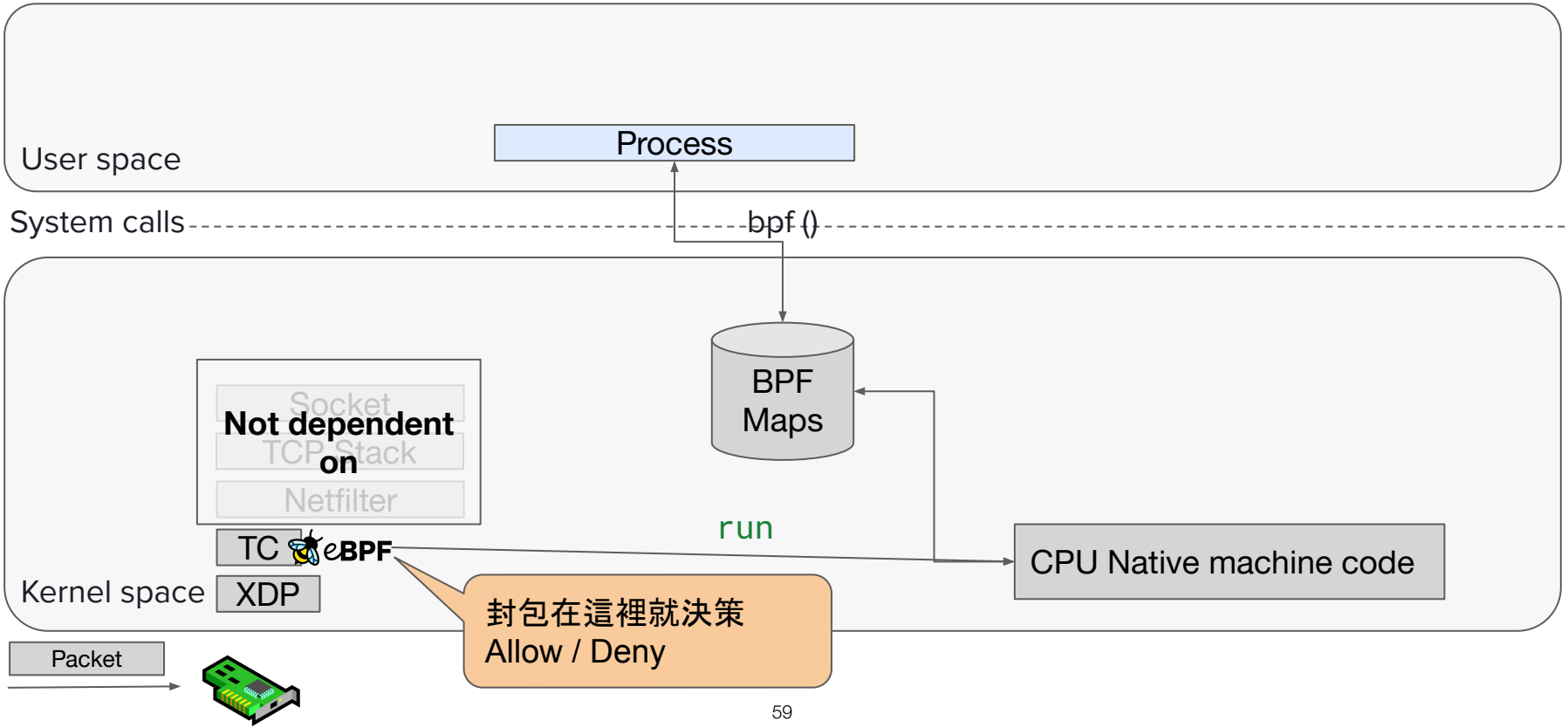
SHOPLINE

先備知識 - Cilium 如何實現 NetworkPolicy

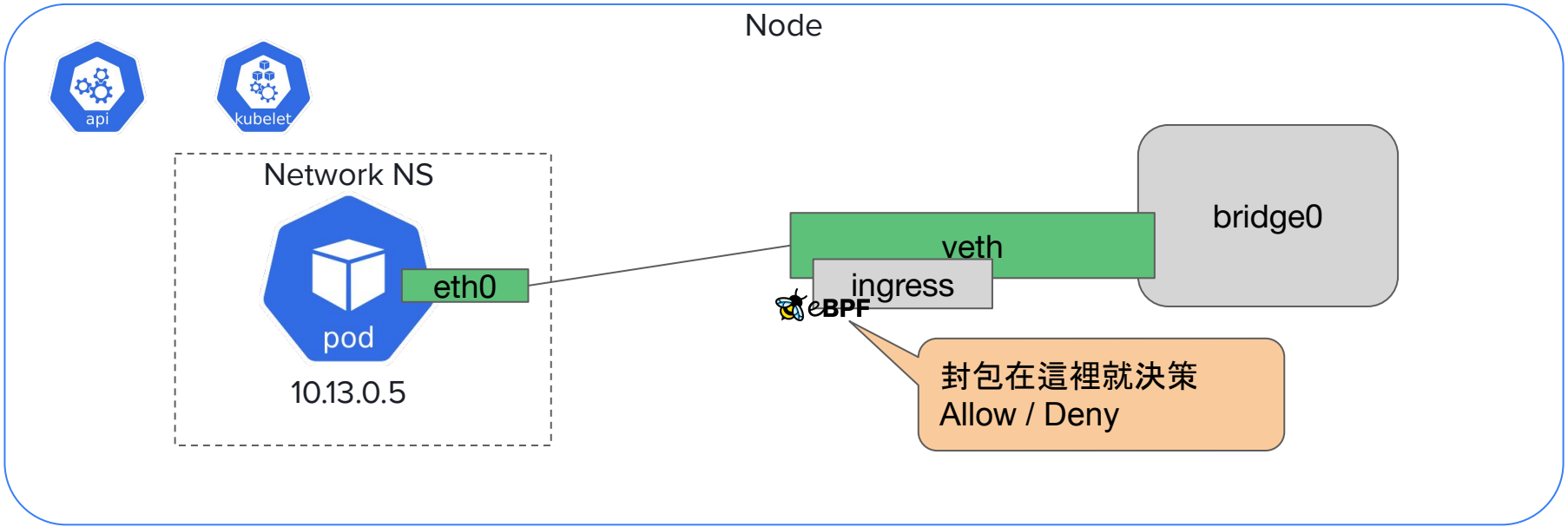


O(NxM)
N: 來源 Identiy 數量
M: 目的 Identity 數量

封包很早就決策！


User space
Process
System calls
bpf ()
BPF
Maps
Socket
Not dependent
on
TCP Stack
Netfilter
TC
eBPF
run
CPU Native machine code
Kernel space
XDP
封包在這裡就決策
Allow / Deny
Packet

封包很早就決策！

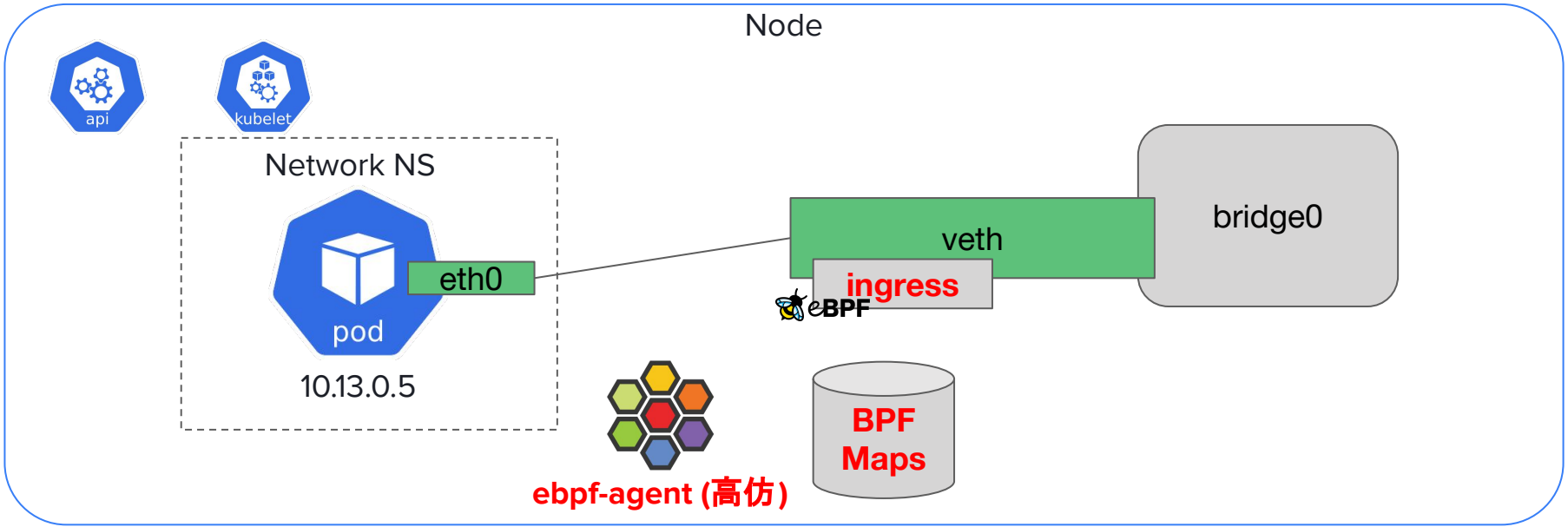

Node
api
kubelet
Network NS
pod
eth0
10.13.0.5
veth
ingress
eBPF
bridge0
封包在這裡就決策
Allow / Deny

預期環境

```
$ kubectl run my-pod
```

預計加入

1. **TC ingress 的 eBPF Prog 修改成 identity-based**
2. **開始使用 eBPF Maps**
3. **實做一個 ebpf-agent**



動手寫 CNI Plugin - 實現 NetworkPolicy

1. 建立 BPF Maps: endpoint_map, policy_map

```
// endpoint_map: 儲存 Pod IP -> Identity (ID) 的對應
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 16384);
    __type(key, __u32);
    __type(value, __u32);
} endpoint_map SEC(".maps");

// policy_map: 儲存允許通訊的來源 ID -> 目的 ID
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 65536);
    __type(key, struct policy_key);
    __type(value, __u8);
} policy_map SEC(".maps");
```

動手寫 CNI Plugin - 實現 NetworkPolicy

1. 建立 BPF Maps: endpoint_map, policy_map
2. 改 BPF Prog，讓邏輯中包含查 BPF Map

```
// 查詢來源 IP 的身份
__u32 *src_id = bpf_map_lookup_elem(&endpoint_map, &src_ip);
if (!src_id)
{
    // 在 map 中找不到，代表是未知的來源，丟棄封包
    return TC_ACT_SHOT;
}

// 查詢目標 IP 的身份
__u32 *dst_id = bpf_map_lookup_elem(&endpoint_map, &dst_ip);
if (!dst_id)
{
    // 在 map 中找不到，代表是未知的目標，丟棄封包
    return TC_ACT_SHOT;
}

// 組合出 policy_map 的查詢 key
struct policy_key pkey = {
    .src_id = *src_id,
    .dst_id = *dst_id,
};

// 查詢是否存在允許的規則
__u8 *action = bpf_map_lookup_elem(&policy_map, &pkey);
if (action && *action == ACTION_ALLOW)
{
    // 找到允許規則，放行封包
    return TC_ACT_OK;
}

// 預設拒絕：沒有找到任何允許規則，因此丟棄封包
return TC_ACT_SHOT;
```

動手寫 CNI Plugin - 實現 NetworkPolicy

1. 建立 BPF Maps: endpoint_map, policy_map
2. 改 BPF Prog, 讓邏輯中包含查 BPF Map
3. 寫一個 ebpf-agent, 要負責
   a. 監聽 Pod, NetworkPolicy
   b. 依照 Labels 分配 Identity
   c. 更新 endpoint_map, policy_map

```
while true; do
  sync_pods
  sync_policies
  gc_unused_identities
  sleep "${INTERVAL}"
done
```

ebpf-agent (高仿)

SHOPLINE

來 Demo 吧 - Network Policy 實踐

```
$ bpftool map dump pinned /sys/fs/bpf/tc/globals/endpoint_map
```

```
$ bpftool map dump pinned /sys/fs/bpf/tc/globals/policy_map
```

SHOPLINE

實戰 Troubleshooting - 想找某 Endpoint 被套哪些 NP

```
root@ip-[redacted]250:/home/cilium# cilium-dbg endpoint get 1433 -o json | jq .[0].status.policy.spec.l4
{
  "egress": [],
  "ingress": [
    {
      "derived-from-rules": [
        [
          "k8s:io.cilium.k8s.policy.derived-from=CiliumNetworkPolicy",
          "k8s:io.cilium.k8s.policy.name=cnp-2025-k8s-summit-allow-frontend",
          "k8s:io.cilium.k8s.policy.namespace=2025-k8s-summit",
          "k8s:io.cilium.k8s.policy.uid=5c43e28a-eed9-4d15-943c-116fa4311b5c"
        ]
      ],
      "rule": "{\"port\":0,\"protocol\":\"ANY\",\"l7-rules\":[{\"\\u0026LabelSelector{MatchLabels:map[string]string{any.app: frontend,k8s.io.kubernetes.pod.namespace: 2025-k8s-summit,},MatchExpressions:[]LabelSelectorRequirement{},}\":null}]}",
      "rules-by-selector": {
        "&LabelSelector{MatchLabels:map[string]string{any.app: frontend,k8s.io.kubernetes.pod.namespace: 2025-k8s-summit,},MatchExpressions:[]LabelSelectorRequirement{},}": [
          [
            "k8s:io.cilium.k8s.policy.derived-from=CiliumNetworkPolicy",
            "k8s:io.cilium.k8s.policy.name=cnp-2025-k8s-summit-allow-frontend",
            "k8s:io.cilium.k8s.policy.namespace=2025-k8s-summit",
            "k8s:io.cilium.k8s.policy.uid=5c43e28a-eed9-4d15-943c-116fa4311b5c"
          ]
        ]
      }
    }
  ]
}
```

實戰 Troubleshooting - 想找某 Endpoint 被套哪些 NP

```
root@ip-[redacted]250:/home/cilium# cilium-dbg bpf policy get 1433
POLICY   DIRECTION   LABELS (source:key[=value])                                                                  PORT/PROTO   PROXY PORT   AUTH TYPE   BYTES   PACKETS   PREFIX
Allow    Ingress     reserved:host                                                                                ANY          NONE         disabled    0       0         0
Allow    Ingress     k8s:app=frontend                                                                             ANY          NONE         disabled    2184    28        0
                     k8s:io.cilium.k8s.namespace.labels.elbv2.k8s.aws/pod-readiness-gate-inject=enabled
                     k8s:io.cilium.k8s.namespace.labels.kubernetes.io/metadata.name=2025-k8s-summit
                     k8s:io.cilium.k8s.policy.cluster=ec-eks-staging
                     k8s:io.kubernetes.pod.namespace=2025-k8s-summit
Allow    Egress      reserved:unknown                                                                             ANY          NONE         disabled    0       0         0
```

SHOPLINE

實戰 Troubleshooting - 想找某 Endpoint 被套哪些 NP

```
# 取得 Endpoint ID
$ kubectl get ciliumendpoint <CEP_NAME> -o yaml | yq .status.id

# 在 cilium-agent，查找該 Endpoint 身上的 policy
$ cilium-dbg endpoint get <EP_ID> -o json | jq .[0].status.policy.spec.l4


# 在 cilium-agent，查找該 Endpoint 的 Policy BPF Map 內容
# 高階版 bpftool map dump pinned /sys/fs/bpf/tc/globals/cilium_policy_<EP_ID>
$ cilium-dbg bpf policy get <EP_ID>
```

總結

痛點/問題	解決方式 /收穫
iptables-based Network Policy 效能慢	• Cilium Identity-Based 模型 • 透過 eBPF 在封包進出早期更早執行決策
Cilium 不熟，但 Cilium 學習曲線很高	• 立即親手實作 Cilium Prototype 揭秘核心原理，建立 Cilium 觀念大局
封包不知道跑哪去了	• Cilium Prototype 建構過程理解 eBPF-based datapath 的排查方向
Network Policy 怎麼不如預期	• 建構 Cilium Prototype 過程展示底層排 查方式 • 使用高度封裝的 cilium-dbg CLI 進行排查

SHOPLINE 召喚神隊友

- 公司持續成長，不論在 Cilium 進階功能或是其他 Cloud Native 的技術都有很多夢想藍圖想邀請你一起完成

Cloud Engineer

Thank you / Q&A

Shiun Chiu
shiun.chiu@shopline.com

本演講原始碼

sh1un/ebpf-based-cni-plugin

附錄 - 補充說明

- Cilium 用到 CO-RE，本演講中沒有特別提及，但是範例 CNI Plugin 也是使用 CO-RE
- Pod IP Assign 是 Cilium Agent 處理，本演講是在 CNI Plugin 處理
- Attach BPF Prog 到 hook point 是 Cilium Agent 處理，本演講是在 CNI
- Cilium 是 per-endpoint Policy Map (cilium_policy_<ep_id>)，本演講是共用一個 Policy Map
- Cilium 維護的 BPF Maps 作用可以參考此文章
- 本演講簡化 NetworkPolicy 的實作，具體 Cilium 在 NetworkPolicy 詳細實作可參考影片
- Cilium 完整教學可參考鐵人賽文章「30 天深入淺出 Cilium：從入門到實戰」