Confluent's Multi-Cloud Journey to Cilium

Pitfalls and Lessons Learned

Nimisha Mehta, Alvaro Aleman

About Us

Alvaro Aleman
Software Engineer
Confluent

Nimisha Mehta
Software Engineer
Confluent

Agenda

1. About Confluent & Cilium
2. Migration Process
3. Issues Encountered
4. Conclusion

What does Confluent do?

- Products related to data streaming and processing

- Confluent cloud is available in AWS, Azure and GCP
- 91 regions
- Lots of Kubernetes infrastructure

Why Cilium?

Security features:

- Transparent encryption
- DNS-based network policies
- Hubble observability

Consistency:

AWS VPC CNI
Azure CNI
GKE CNI

cilium

Enterprise support & SMEs:

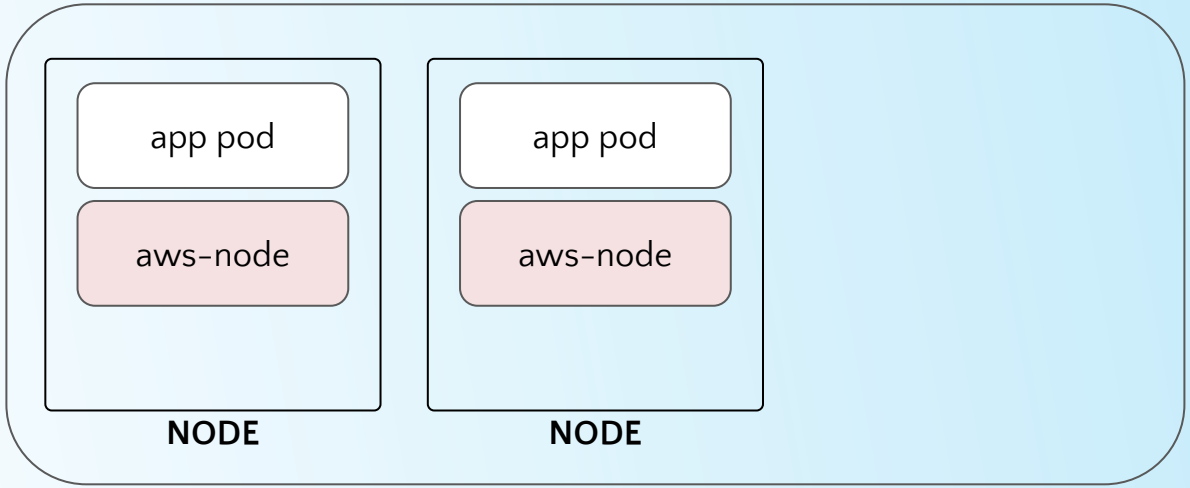
ISOVALENT

Migration process

AWS migration

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: NotIn
         values:
         - cilium
```

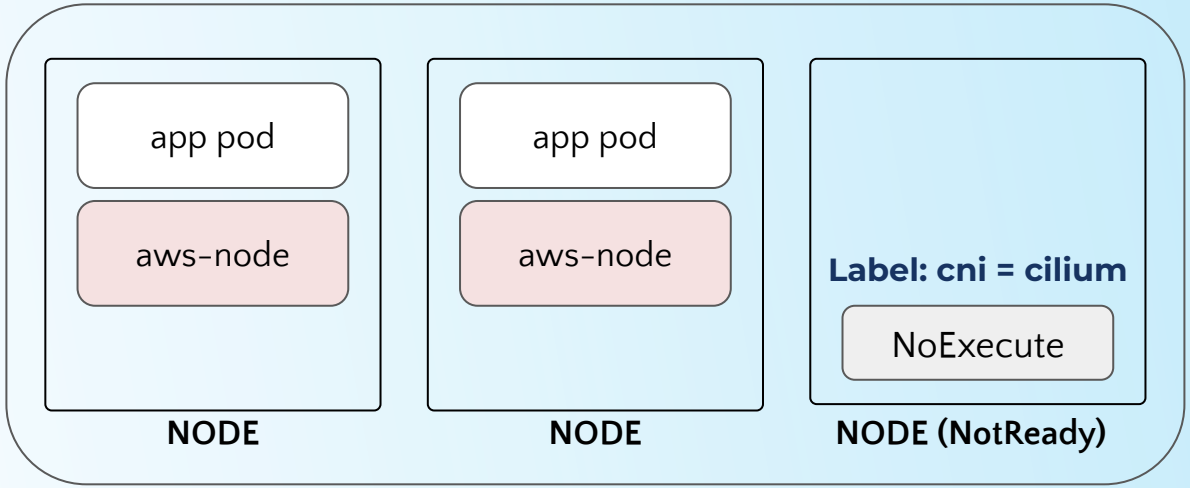
1



AWS migration

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: NotIn
         values:
         - cilium
```

2



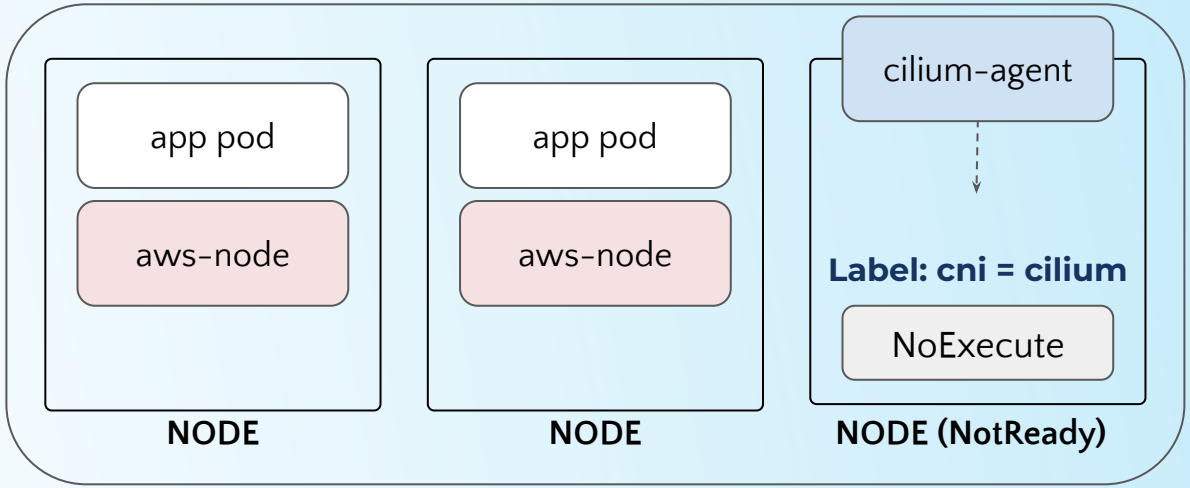
AWS migration

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: NotIn
         values:
         - cilium
```

Install Cilium

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: In
         values:
         - cilium
agentNotReadyTaintKey: "ignore-taint.cluster-autoscaler.
kubernetes.io/node-agent-not-ready"
```

3

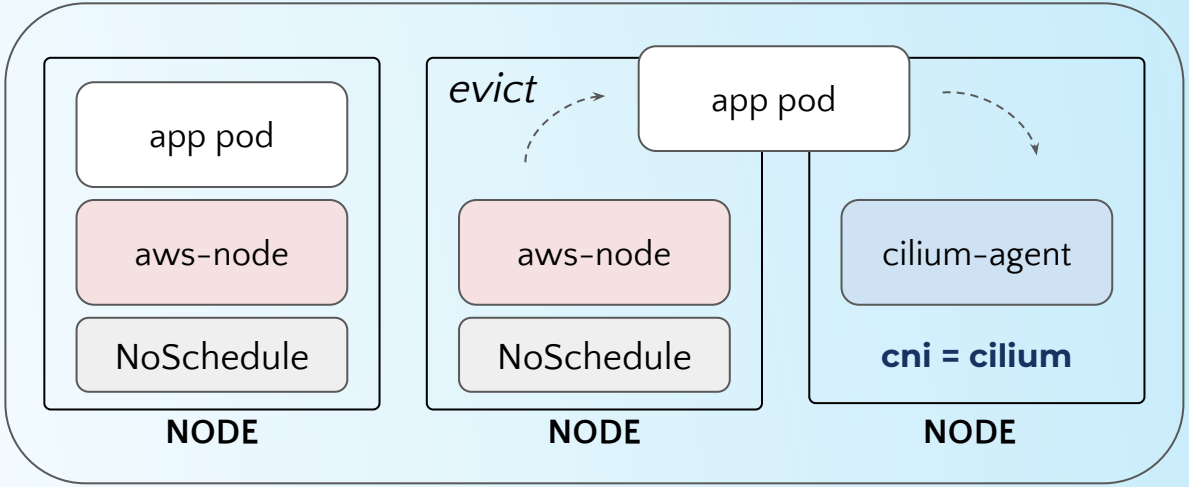


AWS migration

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: NotIn
         values:
         - cilium
```

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: In
         values:
         - cilium
agentNotReadyTaintKey: "ignore-taint.cluster-autoscaler.
kubernetes.io/node-agent-not-ready"
```

4



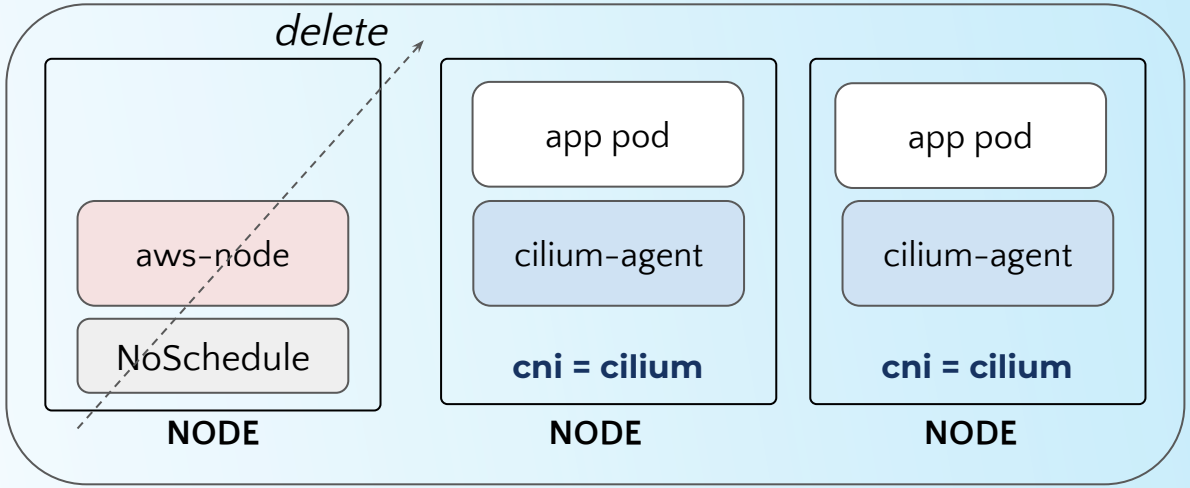
AWS migration

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: NotIn
         values:
         - cilium
```

Delete AWS VPC-CNI

```yaml
affinity:
 nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: cni
         operator: In
         values:
         - cilium
agentNotReadyTaintKey: "ignore-taint.cluster-autoscaler.
kubernetes.io/node-agent-not-ready"
```

5



GCP migration

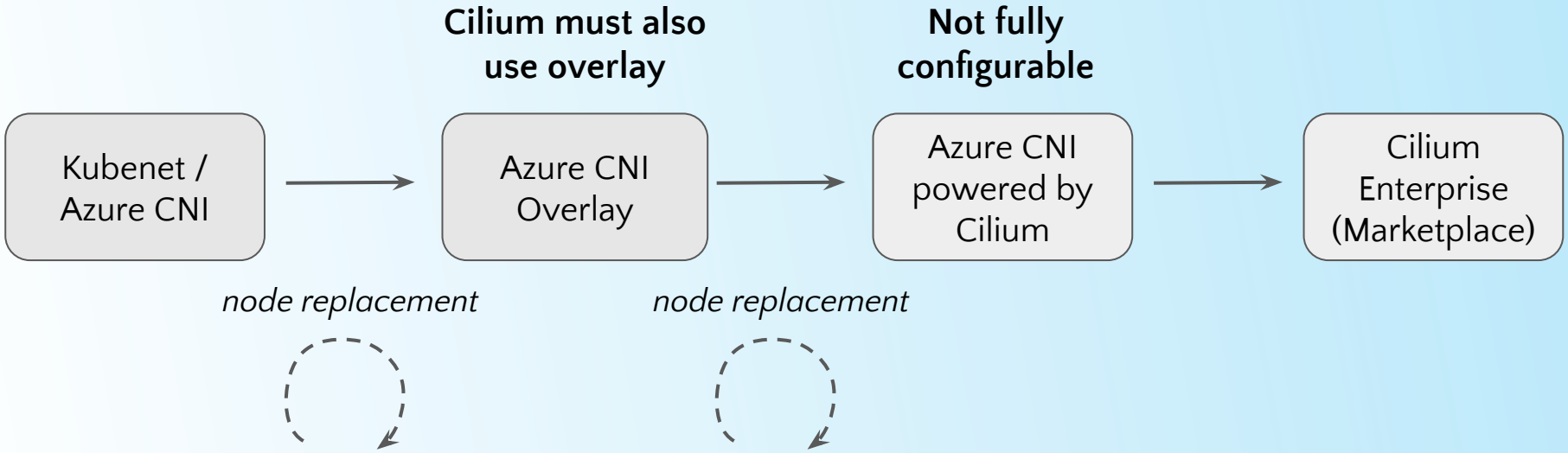
Google Cloud Platform

Setup:

- Dataplane v1 (Dataplane v2 has Cilium out of the box)
- Default GCP CNI always present on nodes – cannot be uninstalled
- Inherited netd daemonset, necessary for some GKE features

netd	Inherited	Enabled by using any of the following: • Intranode visibility • Workload Identity Federation for GKE • IPv4/IPv6 dual-stack networking

Azure migration

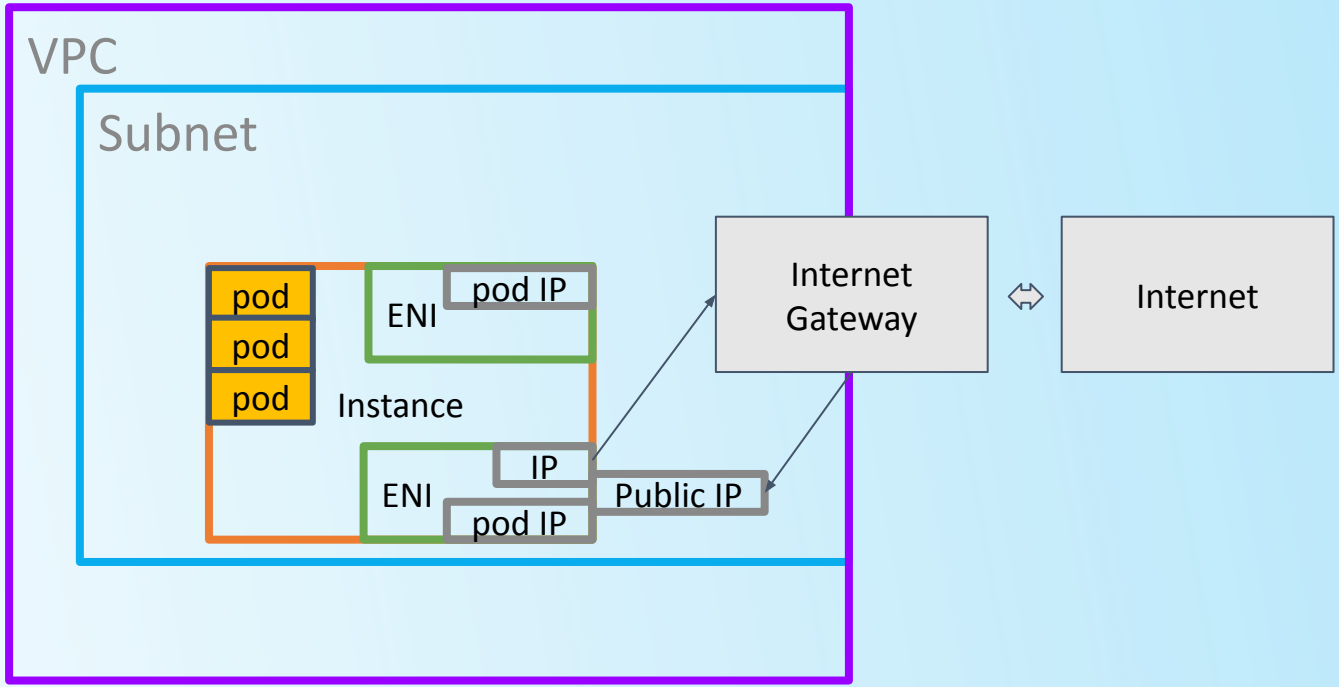


Issues encountered

Issue 1: AWS Reverse Path Filtering

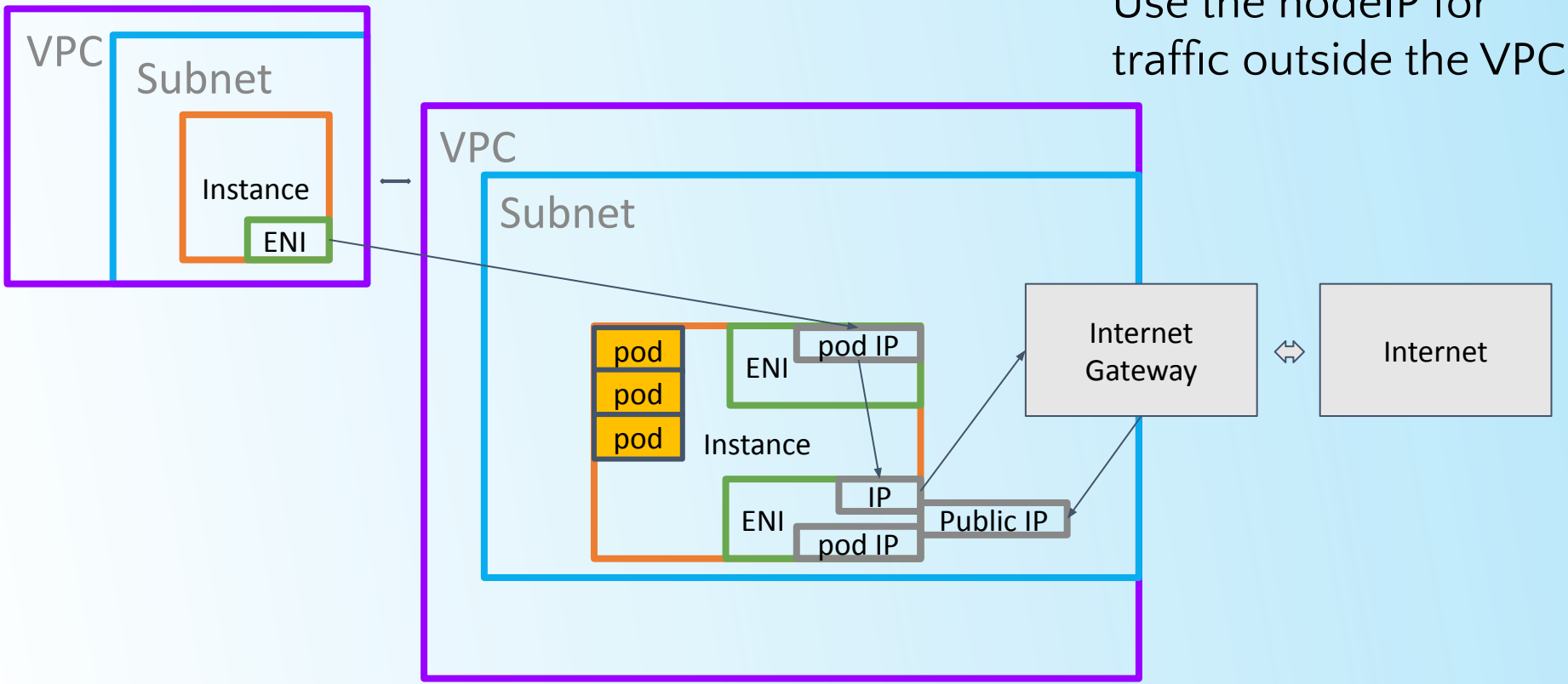
AWS Reverse Path Filtering

Egress masquerading:
Use the nodeIP for traffic outside the VPC



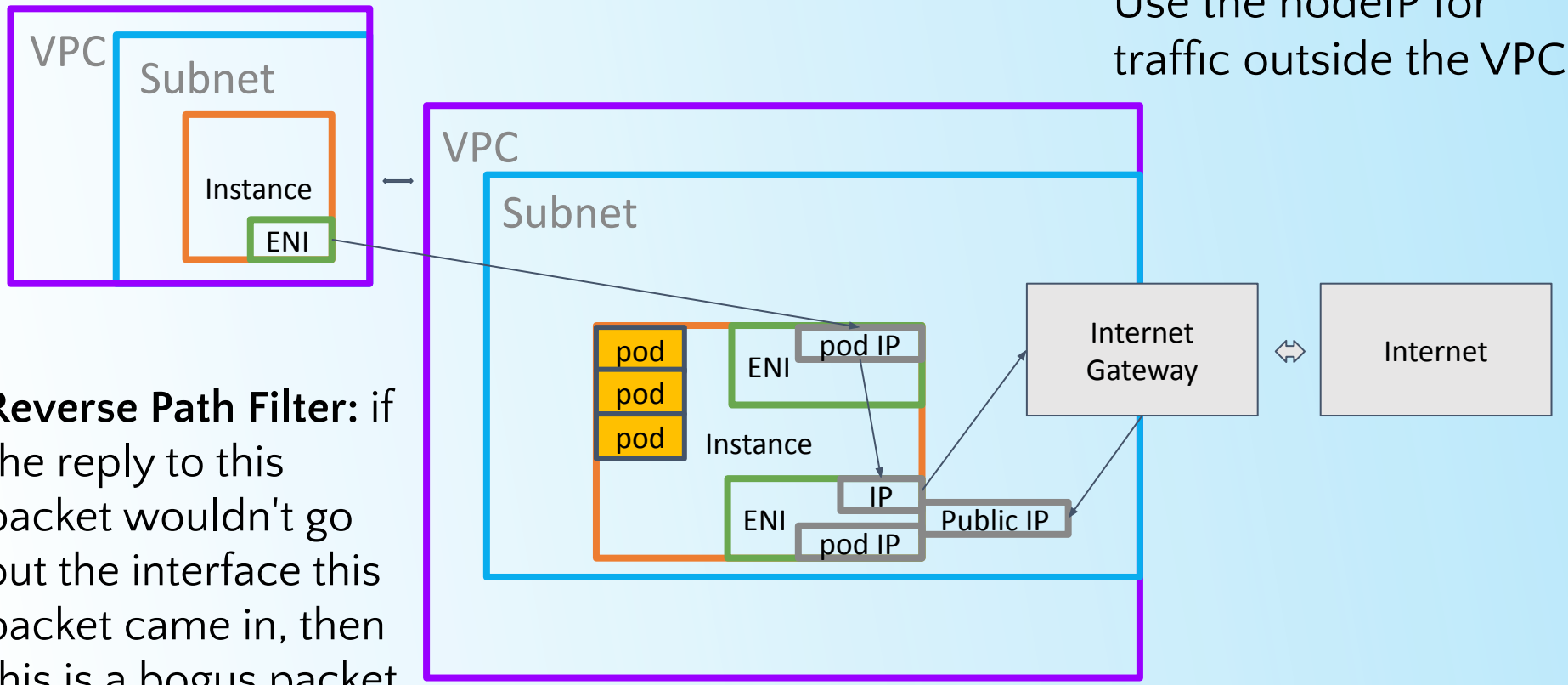
AWS Reverse Path Filtering

Egress masquerading:
Use the nodeIP for traffic outside the VPC



AWS Reverse Path Filtering

Egress masquerading: Use the nodeIP for traffic outside the VPC



Reverse Path Filter: if the reply to this packet wouldn't go out the interface this packet came in, then this is a bogus packet and should be ignored

AWS Reverse Path Filtering

Workaround: Use nodePort service

Drawbacks:

- More complicated
- Health signal gets diluted if multiple pods for the same service are on a node

Issue 2: GCP FQDN policy conflict

GCP FQDN policy conflict

Cilium Policy
(default enforcement mode)

➔ Breaks all DNS within the cluster

Hubble: no policy denials, but DNS works on deleting the policy


```yaml
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: allow-some-egress
spec:
  egress:
  - toFQDNs:
    - matchPattern: "*.internal.com."
...
  - toEndpoints:
    - matchLabels:
        k8s-app: kube-dns
    toPorts:
    - ports:
      - port: "53"
        protocol: ANY
      rules:
        dns:
        - matchPattern: '*'
  endpointSelector:
    matchLabels: {}
```

DNS breaks when intercepted by Cilium's DNS proxy? 🤔

GCP FQDN policy conflict

```
diff --git a/bad b/good

index 07f68d1..c4c18fa 100644

--- a/bad
+++ b/good

@@ -1,22 +1,10 @@

-net.ipv4.conf.all.src_valid_mark = 1
+net.ipv4.conf.all.src_valid_mark = 0
```

Bad node has *src_valid_mark* set to 1

- **src_valid_mark:** Linux networking configuration & source address validation mechanism
- When enabled, the kernel performs additional validation on the source address of a packet
- When going through the **Cilium DNS proxy**, the kernel considers DNS packets **invalid**, and **drops** them

GCP FQDN policy conflict

Unsetting the value manually...

```
$ sysctl -w net.ipv4.conf.all.src_valid_mark=0
```

And DNS starts working!! 🎉

Until...it gets set again by something

```
$ sysctl net.ipv4.conf.all.src_valid_mark
net.ipv4.conf.eth0.src_valid_mark = 1
```

Which process is making the relevant syscall?

It's netd 🙃

```
root        4145  0.0  0.2 1262820 33276 ?
Ssl  Jun26   0:21 /netd --enable-source-valid-
mark=true ...
```

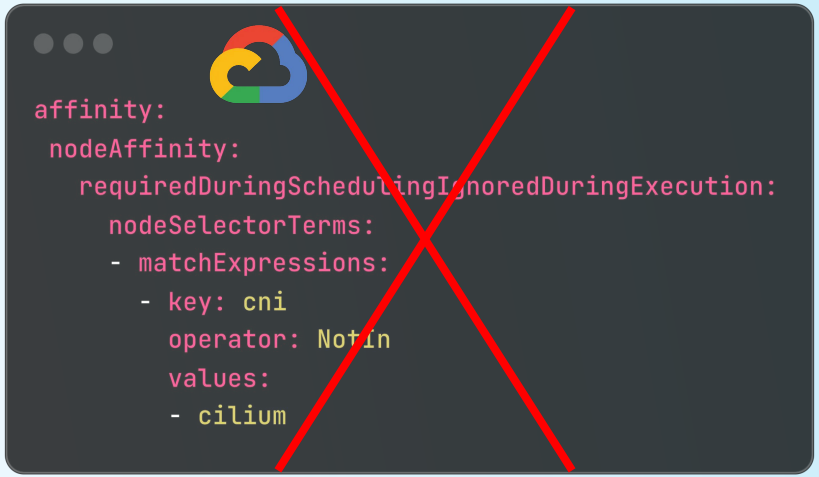
GCP FQDN policy conflict

- Finally traced to GKE feature – *intranode visibility*
- Disabled in favor of Cilium FQDN-based policies
- Suggested as a roadmap item for Cilium to monitor and warn on unexpected *src_valid_mark* value

Issue 3: GCP Pod IPs

GCP Pod IPs

Not possible: CNI is baked in



GCP Pod IPs

```
NAME                    IP             STATUS    AGE
demo-5999fbb588-jt6s8   172.16.0.66    Running   2024-10-11T17:52:47Z
demo-5999fbb588-w9ms5   172.16.0.66    Running   2024-10-11T17:46:10Z
```

GCP Pod IPs

...Why?

- Node becomes “Ready” so other pods might come up before Cilium
- No other CNI existing implicitly prevents this on other clouds
- ...use a taint?
- Workaround: Controller that deletes unmanaged pods

Issue(s) 4: Misconfiguration

Issues arising out of misconfiguration

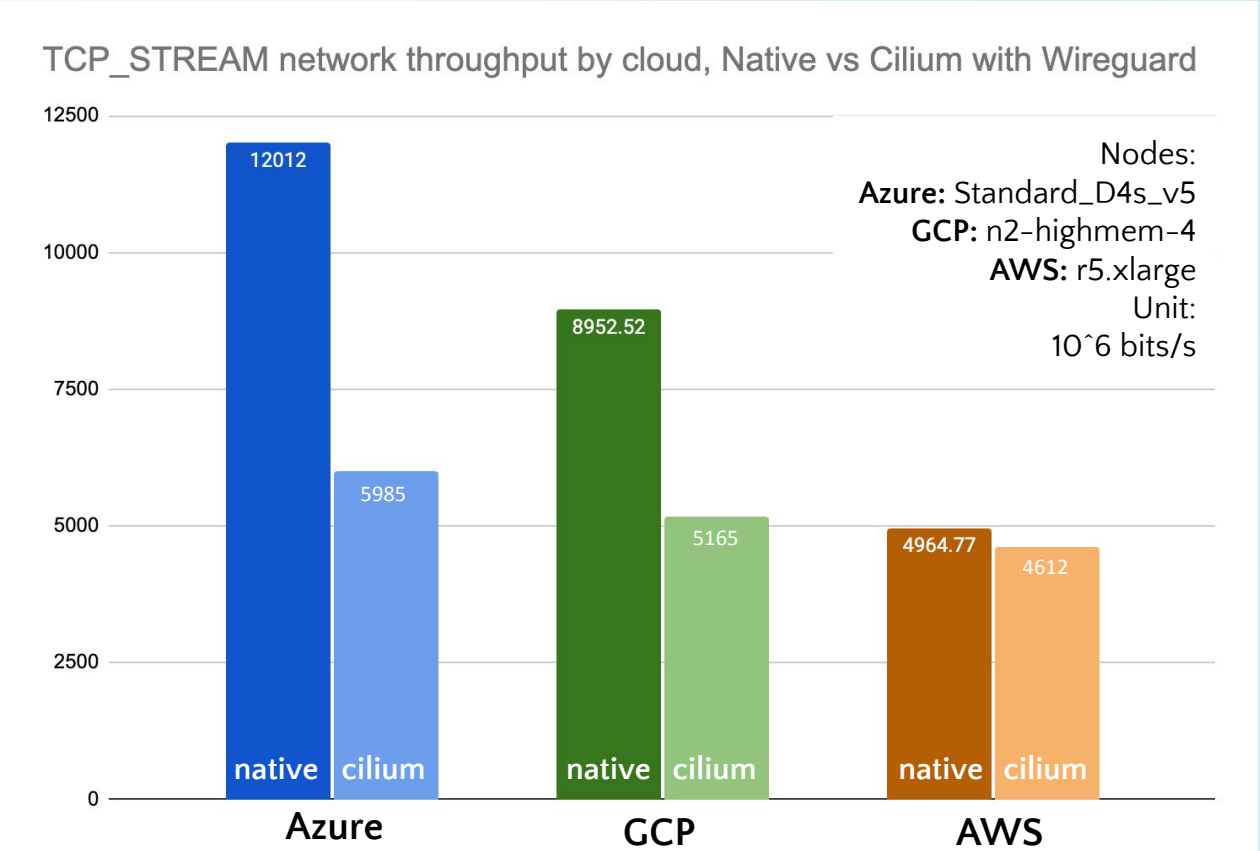
- As of October 2024, 800+ configurable Cilium Helm values
- Unique config based on network setup & cloud provider
- Example: Pod Hubble metrics cardinality explosion when not using required labels

```
flows-to-world:sourceContext=namespace|dns|ip;destinationContext=na
mespace|dns|ip;port;
```

VS

```
flows-to-world:sourceContext=namespace|dns|ip;destinationContext=na
mespace|dns|ip;port;labelsContext=source_namespace,source_pod
```

Network overhead of encryption



*cilium with Wireguard TE

Encryption enabled but not actually happening?

```
root@gke-test-00-mainpool--v2-25dddcb3-k0sj:/home/cilium# tcpdump -n -i
cilium_wg0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on cilium_wg0, link-type RAW (Raw IP), snapshot length 262144 bytes

^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

- Traced to incorrect Helm value: native routing CIDR
- Pod traffic outside incorrect CIDR range gets SNAT'ed as node traffic
- Can bypass encryption or even network policy enforcement!

Issue 5: Azure hostPorts

Azure hostPorts

What happened?

- Upgraded the Kubernetes version of a number of Azure clusters
- Workloads that use a hostPort are sometimes not reachable
- Restarting the Cilium agent helps

Azure hostPorts

```
1  Chain PREROUTING (policy ACCEPT)
2  target          prot opt source               destination
3  KUBE-SERVICES   all  --  anywhere             anywhere
4  CILIUM_PRE_nat all  --  anywhere             anywhere
```

```
1  Chain PREROUTING (policy ACCEPT)
2  target          prot opt source               destination
3  CILIUM_PRE_nat all  --  anywhere             anywhere
4  KUBE-SERVICES   all  --  anywhere             anywhere
```

Azure hostPorts

```yaml
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - env:
    - name: KUBERNETES_SERVICE_HOST
      value: k8s-name.hcp.region.azmk8s.io
```

Azure hostPorts

```
k exec cilium-xbljt -- bash -c 'echo $KUBERNETES_SERVICE_HOST'

k8s-name.hcp.region.azmk8s.io
```

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - env:
    - name: KUBERNETES_SERVICE_HOST
      value: k8s-name.hcp.region.azmk8s.io
```

VS

```
k exec -n some-namespace a-pod -- bash -c 'echo $KUBERNETES_SERVICE_HOST'

10.253.0.1
```

Azure hostPorts

```
k exec cilium-xbljt -- bash -c 'echo $KUBERNETES_SERVICE_HOST'

k8s-name.hcp.region.azmk8s.io
```

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - env:
    - name: KUBERNETES_SERVICE_HOST
      value: k8s-name.hcp.region.azmk8s.io
```

VS

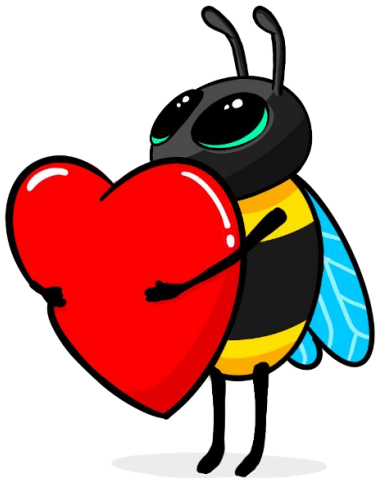
```
k exec -n some-namespace a-pod -- bash -c 'echo $KUBERNETES_SERVICE_HOST'

10.253.0.1
```

➔ Block Cilium startup until kube-proxy rule exists

Journey So Far

- Successfully migrated production clusters to Cilium
- Cilium is becoming our go-to tool for securing cluster workloads
- Realistically, some issues are discovered only in production
- Cilium has great community & tooling which makes things very accessible for end-users



Thank you!