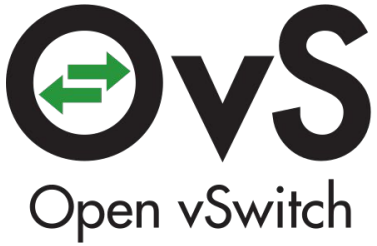
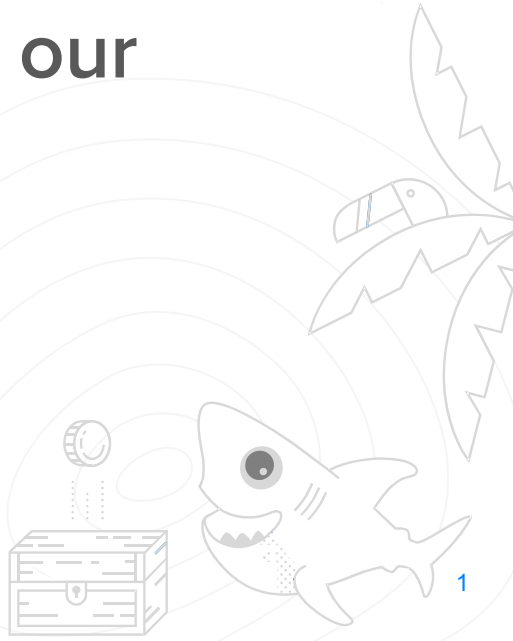


How virtual machines access our metadata service

Nick Bouliane

Open vSwitch conference 2021

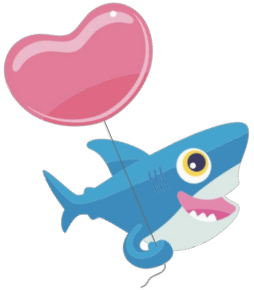


What we will see...

- ❏ What is a metadata service
- ❏ The datapath to access the metadata service
- ❏ The flow needed for the requests to succeed
- ❏ BONUS: code sample to show how to programmatically create this datapath (not necessarily production ready)

DigitalOcean

- Cloud hosting company
- > 20 000 hypervisors running OpenvSwitch
- 13 Data centers around the world
  - NYC1, NYC2, NYC3: New York City, United States
  - AMS2, AMS3: Amsterdam, the Netherlands
  - SFO1, SFO2, SFO3: San Francisco, United States
  - SGP1: Singapore
  - LON1: London, United Kingdom
  - FRA1: Frankfurt, Germany
  - TOR1: Toronto, Canada
  - BLR1: Bangalore, India

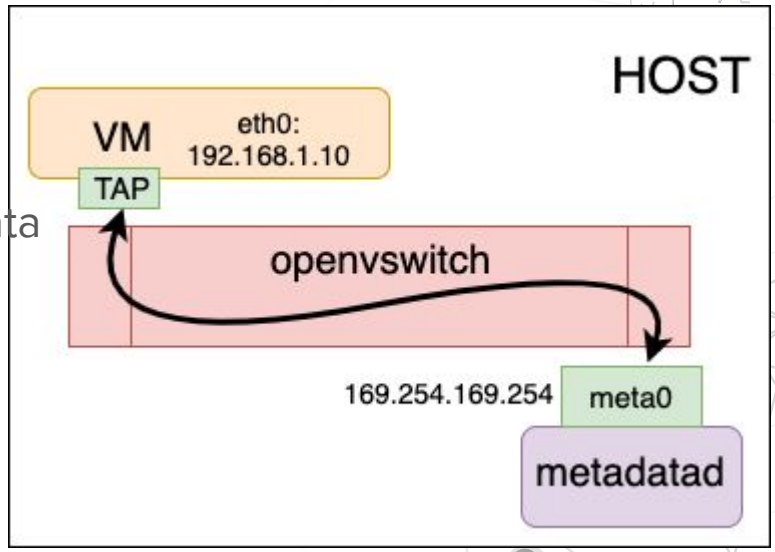


The metadata service

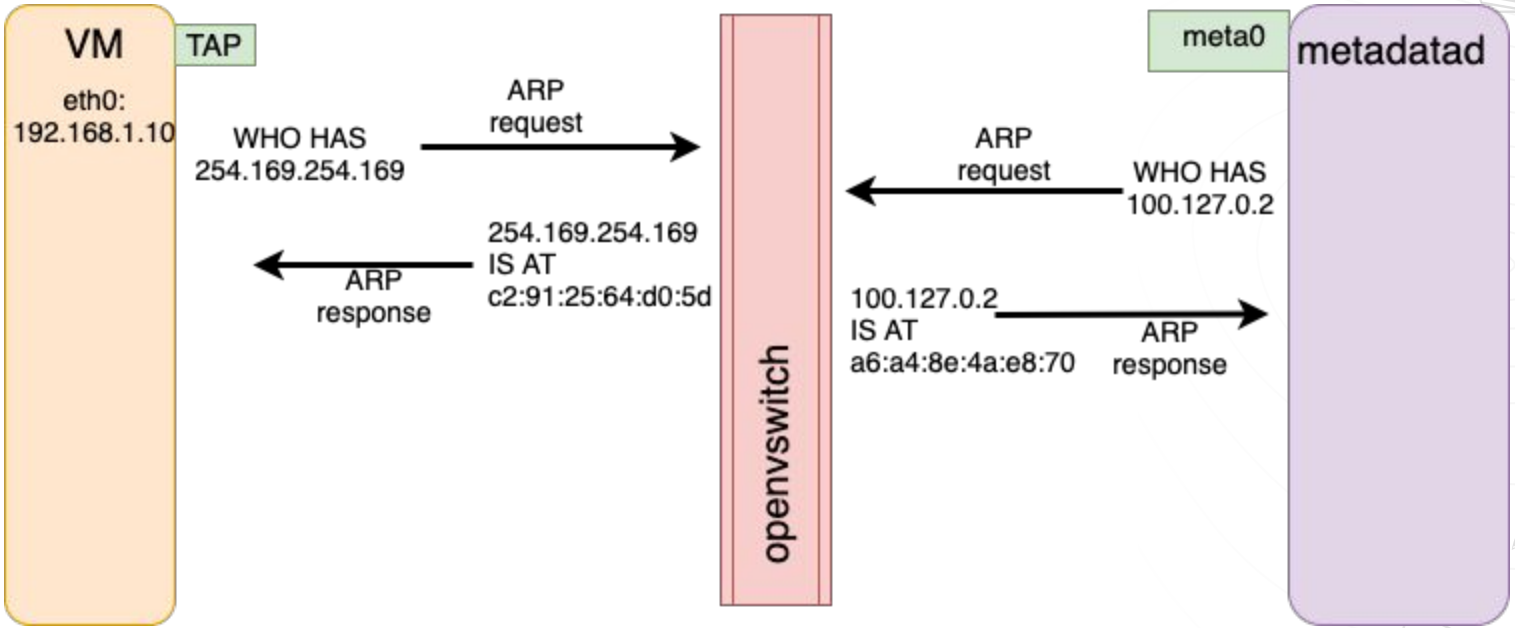
- An HTTP server
- 169.254.169.254
- What kind of data ?
  - Virtual machine properties (hostname, public keys, region, etc)
  - Network interfaces (gateway, address, netmask, etc)
  - Floating IPs (status, address, etc)
  - DNS (nameservers, etc)
- Who is asking for the data

The datapath to access the metadata service

- VM TAP plugged into the switch
- meta0 is a an internal port (openvswitch driver)
- meta0 is plugged into the switch
- metadatad listen 169.254.169.254:80
- ip rule and table
  - from 169.254.169.254 lookup metadata
  - default dev meta0 scope link
- ARP responder
- NAT



The flows - ARP responder



The flows - ARP responder

- Flow to respond to the ARP request sent by the virtual machine

arp,in_port=tapext7889371,dl_src=a6:a4:8e:4a:e8:70,arp_tpa=169.254.169.254,arp_op=1
actions=load:0x2->NXM_OF_ARP_OP[],mod_dl_src:c2:91:25:64:d0:5d,mod_dl_dst:a6:a4:8e:4a:e8:70,load:0xc2912564d05d->NXM_NX_ARP_SHA[],load:0xa6a48e4ae870->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xa9fea9fe->NXM_OF_ARP_SPA[],IN_PORT

Internet Protocol (IPv4) over Ethernet ARP packet

Octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	

8	Sender hardware address (SHA) (first 2 bytes)
10	(next 2 bytes)
12	(last 2 bytes)
14	Sender protocol address (SPA) (first 2 bytes)
16	(last 2 bytes)
18	Target hardware address (THA) (first 2 bytes)
20	(next 2 bytes)
22	(last 2 bytes)
24	Target protocol address (TPA) (first 2 bytes)
26	(last 2 bytes)

The flows - ARP responder

```
// Respond to ARP requests for metadata from droplets with meta0's MAC
//   - Inbound from droplet
//   - ARP request
//   - ARP TPA matching metadata IP
//   - MAC address of droplet
// Action:
//   - Rewrite ARP OP to reply
//   - Set source and destination ethernet to meta0 and the droplet
//   - Set "arp_sha" to meta0's MAC
//   - Set "arp_tha" to droplet's MAC
//   - Set "arp_tpa" to the IP used as arp_spa in the request
//   - Set "arp_spa" to the metadata IP
//   - Output the packet on the original input port
```

```
{
    Priority: 4030,
    Protocol: ovs.ProtocolARP,
    InPort:   droplet.PortID,
    Matches: []ovs.Match{
        ovs.ARPOperation(arpOpRequest),
        ovs.ARPTargetProtocolAddress(metadataIP),
        ovs.DataLinkSource(r.HardwareAddr.String()),
    },
    Table: tableARPResponder,
    Actions: []ovs.Action{
        ovs.Load("0x2", "OXM_OF_ARP_OP[]"),

        ovs.ModDataLinkSource(r.MetadataHardwareAddr),
        ovs.ModDataLinkDestination(r.HardwareAddr),

        ovs.SetField(r.MetadataHardwareAddr.String(), "arp_sha"),
        ovs.SetField(r.HardwareAddr.String(), "arp_tha"),
        ovs.Move("OXM_OF_ARP_SPA[]", "OXM_OF_ARP_TPA[]"),
        ovs.SetField(metadataIP, "arp_spa"),

        ovs.InPort(),
    },
},
```

The flows - ARP responder

- Flow to respond to the ARP request sent by the metadata (host)

arp,in_port=meta0,dl_src=c2:91:25:64:d0:5d,arp_spa=169.254.169.254,arp_tpa=100.127.2.90,arp_op=1
actions=load:0x2->NXM_OF_ARP_OP[],mod_dl_src:a6:a4:8e:4a:e8:70,mod_dl_dst:c2:91:25:64:d0:5d,load:0
xa6a48e4ae870->NXM_NX_ARP_SHA[],load:0xc2912564d05d->NXM_NX_ARP_THA[],load:0x647f025a->N
XM_OF_ARP_SPA[],load:0xa9fea9fe->NXM_OF_ARP_TPA[],IN_PORT

Internet Protocol (IPv4) over Ethernet ARP packet

Octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	

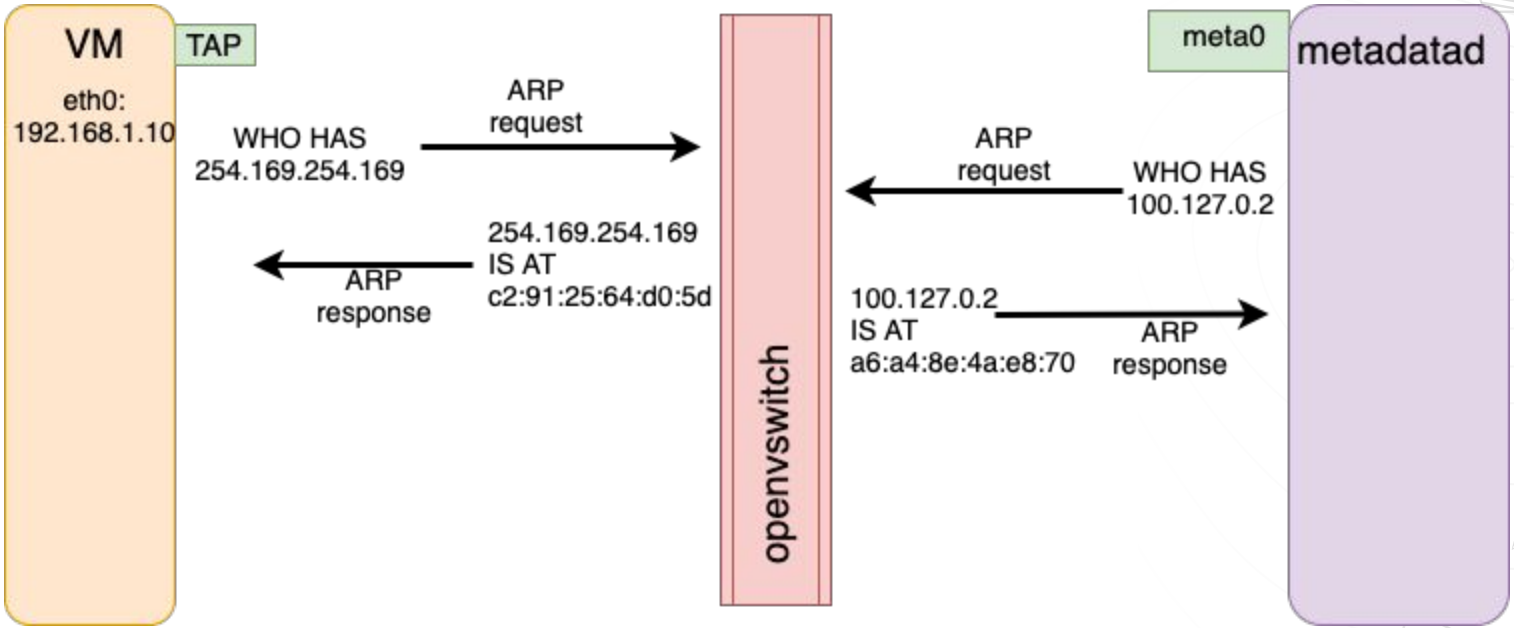
8	Sender hardware address (SHA) (first 2 bytes)
10	(next 2 bytes)
12	(last 2 bytes)
14	Sender protocol address (SPA) (first 2 bytes)
16	(last 2 bytes)
18	Target hardware address (THA) (first 2 bytes)
20	(next 2 bytes)
22	(last 2 bytes)
24	Target protocol address (TPA) (first 2 bytes)
26	(last 2 bytes)

The flows - ARP responder

```
// Respond to ARP requests for droplets from metadata with the droplet's MAC
//   - Inbound from the metadata service
//   - ARP query from metadata IP
//   - ARP TPA matching Droplet's HV local address for metadata
//   - MAC address of metadata service
// Action:
//   - Rewrite ARP OP to reply

//   - Set source and destination ethernet to droplet and meta0

//   - Set "arp_sha" to droplet's MAC
//   - Set "arp_tha" to meta0's MAC
//   - Set "arp_spa" to droplet's HV-local address for metadata
//   - Set "arp_tpa" to the metadata IP

//   - Output the packet on the original input port
```

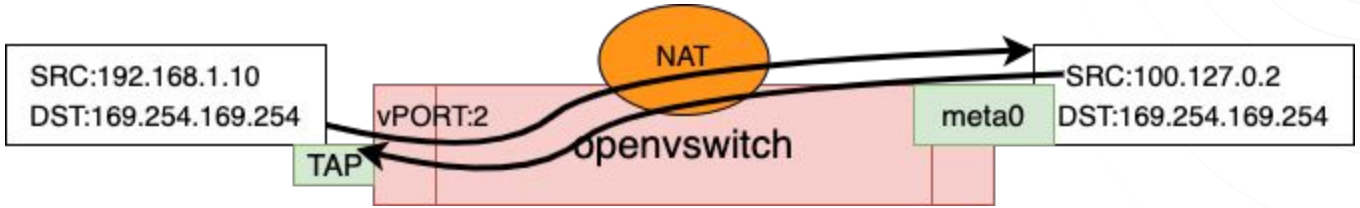
```
{
    Priority: 4030,
    Protocol: ovs.ProtocolARP,
    InPort:   meta.PortID,
    Matches: []ovs.Match{
        ovs.ARPOperation(arpOpRequest),
        ovs.ARPTargetProtocolAddress(hvLocalAddr),
        ovs.ARPSourceProtocolAddress(metadataIP),
        ovs.DataLinkSource(r.MetadataHardwareAddr.String()),
    },
    Table: tableARPResponder,
    Actions: []ovs.Action{
        ovs.Load("0x2", "OXM_OF_ARP_OP[]"),

        ovs.ModDataLinkSource(r.HardwareAddr),
        ovs.ModDataLinkDestination(r.MetadataHardwareAddr),

        ovs.SetField(r.HardwareAddr.String(), "arp_sha"),
        ovs.SetField(r.MetadataHardwareAddr.String(), "arp_tha"),
        ovs.SetField(hvLocalAddr, "arp_spa"),
        ovs.SetField(metadataIP, "arp_tpa"),

        ovs.InPort(),
    },
},
```

The flows - ARP responder



The flows - Network Address Translation

- We use a subnet E.g 100.127.0.0/16
- map the 16 bits OVS openflow port
  - ofPort 2 IP ⇒ 100.127.0.2
- Metadata service translate ofPort to Virtual Machine ID
  - ofPort 2 ⇔ VM ID 1234



The flows - Network Address Translation

- The flow that translate the source address of the packet from the VM

tcp,in_port=tapext7889371,dl_src=a6:a4:8e:4a:e8:70,nw_dst=169.254.169.254,tp_dst=80 actions=ct(commit,zone=602,nat(src=100.127.0.2),exec(load:0x7861db->NXM_NX_CT_MARK[])),resubmit(,25)

```
// Classify IPv4 traffic:
//   - Inbound from a droplet port
//   - TCP/80 traffic
//   - Destination is 169.254.169.254 (metadata IP address)
//   - MAC address of Droplet
// Action:
//   - Source NAT with hvLocalAddr and commit to conntrack
//   - Resubmit to L2 rewrite table
```

```
{
    Priority: 4010,
    Protocol: ovs.ProtocolTCPv4,
    InPort:   droplet.PortID,
    Matches: []ovs.Match{
        ovs.NetworkDestination(metadataIP),
        ovs.TransportDestinationPort(80),
        ovs.DataLinkSource(r.HardwareAddr.String()),
    },
    Actions: []ovs.Action{
        ovs.ConnectionTracking(fmt.Sprintf("zone=%d,commit,nat(src=%s),exec(set_field:%d->ct_mark)",
            droplet.PortID,
            hvLocalAddr,
            r.DropletID,
        )),

        ovs.Resubmit(0, tableL2Rewrite),
    },
},
```

The flows - Network Address Translation

- The opposite direction, send packet to conntrack

tcp,metadata=0,in_port=meta0,dl_dst=a6:a4:8e:4a:e8:70,nw_src=169.254.169.254,nw_dst=100.127.2.90,tp_src=80 actions=load:0x1->OXM_OF_METADATA[],ct(table=65,zone=602,nat)

```
// Match
//   - Source IP is metadata service IP
//   - Destination IP is Droplet HV Local IP
//   - Destination MAC is Droplet
//   - Source TCP port is 80
//   - CT state is not tracked (for bionic)
// Action:
//   - Set the metadata field to mark that this rule have
//     match and sent the packet through conntrack
//   - Send the packet through CT and recirculate to
//     table forwarding
```

```
{
    Priority: 4012,
    Protocol: ovs.ProtocolTCPv4,
    InPort:   meta.PortID,
    Matches: []ovs.Match{
        ovs.Metadata(0),
        ovs.NetworkSource(metadataIP),
        ovs.NetworkDestination(hvLocalAddr),
        ovs.DataLinkDestination(r.HardwareAddr.String()),
        ovs.TransportSourcePort(80),
    },
    Table: tableForwarding,
    Actions: []ovs.Action{
        ovs.SetField("0x1", "metadata"),
        ovs.ConnectionTracking(fmt.Sprintf("table=%d,zone=%d,nat",
            tableForwarding,
            droplet.PortID)),
    },
},
```

The flows - Network Address Translation

tcp,ct_state=+est+trk,metadata=0x1,in_port=meta0,dl_dst=a6:a4:8e:4a:e8:70,nw_src=169.254.169.254,tp_src=80 actions=output:tapext7889371

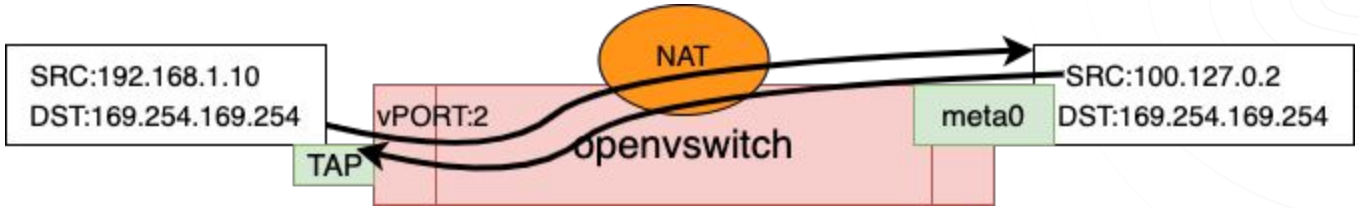
```
// Match
//    - Source IP is metadata service IP
//    - Destination MAC is Droplet
//    - Source TCP port is 80
//    - CT state established and tracked
// Action:
//    - Output to droplet port
```

```
{
    Priority: 4010,
    Protocol: ovs.ProtocolTCPv4,
    InPort:   meta.PortID,
    Matches: []ovs.Match{
        ovs.Metadata(1),
        ovs.NetworkSource(metadataIP),
        ovs.DataLinkDestination(r.HardwareAddr.String()),
        ovs.TransportSourcePort(80),
        ovs.ConnectionTrackingState(
            ovs.SetState(ovs.CTStateEstablished),
            ovs.SetState(ovs.CTStateTracked)),
    },
    Table: tableForwarding,
    Actions: []ovs.Action{
        ovs.Output(droplet.PortID),
    },
},
```

Accessing the metadata service

Example:

curl -s http://169.254.169.254/metadata/v1/interfaces/public/0/ipv4/address



Conclusion

- what is a metadata service
- The datapath to access the metadata service
  - ARP responder flows
  - NAT flows
  - how we carry the ofPort information via the IP address
- code sample using go-openvswitch

DigitalOcean