Using OVN Interconnect for scaling (OVN) Kubernetes deployments

Numan Siddique
Dumitru Ceara
K8S CNI plugin

Uses OVN and OVS

OVN Community project - [https://github.com/ovn-org/ovn-kubernetes](https://github.com/ovn-org/ovn-kubernetes)
Present OVN-Kubernetes Architecture (raft)
What’s the problem now?

- OVN southbound database
  - becomes a bottleneck as the number of nodes increase.
  - Raft issues - split brain, frequent leadership transfers.
- ovsdb-server is single threaded.
- ovn-northd does not process changes incrementally and its complexity is $O(N \times M)$ with $N$ nodes and $M$ services (likely with a constant $> 1$)
• **OVN Interconnection** is a feature of OVN
• Allows independent OVN deployments to be interconnected by OVN managed geneve tunnels.
• Requires
  ○ Global interconnect databases accessible from each deployment
  ○ “ovn-ic” service running on each deployment.
• ovn-ic connects to global ic databases and also to its OVN Northbound and Southbound databases.
• It creates transit switch in OVN Northbound database for interconnection.
• Please refer to this presentation from Han for more information
Motivation to use OVN IC

- Address scale requirements.
- Avoid worker nodes communicating to the NB/SB databases running in central/master nodes
Proposed OVN-Kubernetes Architecture (IC)
Can OVN Interconnect (IC) solve the scale issues?

1. OVN component communication is now isolated per node - no network traffic for clients (ovn-controllers) to talk to database servers (SBDB)

2. Only a single client per database - eliminated current bottleneck of SBDB cannot scale with as n clients increase

3. Smaller per node database size - northd CPU pressure is reduced and database sizes are smaller since a node only needs a subset of the data

4. No more Raft with every node having its own database - eliminates a source of complexity and severe bugs
OVN IC Technical Overview
Proposed Interconnect support in OVN-K8S

- No native interconnect OVN databases or “ovn-ic” service required
- Interconnect functionality is added in ovn-kubernetes using zones
What is a zone?

- A zone is an independent OVN deployment
- A K8s deployment can have one or more zones.
- A zone can have one or more kubernetes nodes.
- Each kubernetes node is assigned to a zone.
- Each zone will run its own ovnkube-master(s) (multiple ovnkube-masters for HA)
Example deployment - 3 master and 5 worker nodes
OVN-K8S network topology (centralized)

- ovn_cluster_router distributed
- join_switch distributed
OVN-K8S network topology (IC)

- ovn_cluster_routers not distributed anymore
- join_switch not distributed anymore
- transit-switch distributed
1. ovnkube-master creates POD4 logical switch port in NB.
2. ovnkube-node (node2) creates POD4 veth in OVS.
3. ovn-northd creates SB port binding.
4. ovn-controller (node2) claims the port and installs all required openflows.

Adding a POD (centralized)
1. ovnkube-master-2 creates POD4 logical switch port in NB-2
2. ovnkube-node (node2) creates POD4 veth in OVS
3. ovn-northd-2 creates SB-2 port binding
4. ovn-controller (node2) claims the port and installs all required openflows
1. ovnkube-master creates a load balancer for the service with backends POD2 and POD4. This is applied to the node switches and node GRs.

2. ovn-northd creates SB load balancer and relevant logical flows.

3. ovn-controllers process SB updates and installs all required openflows.
Adding a service (IC)

1. ovnkube-masters create a load balancer for the service with backends POD2 and POD4. This is applied to the node switches and node GRs.

2. ovn-northds create SB load balancer and relevant logical flows.

3. ovn-controllers process SB updates and installs all required openflows.
1. ovnkube-master creates an ACL for the network policy. The ACL refers to the port group containing all selected pods, PG1=(POD2, POD4) and to the address set containing the pods’ IPs AS1 = (POD2-IP, POD4-IP). The ACL is (implicitly) applied to the node switches.

2. ovn-northd creates SB relevant logical flows

3. ovn-controllers process SB updates and installs all required openflows
1. **ovnkube-masters** create an ACL for the network policy. The ACL refers to the port group containing all locally selected pods, $PG1 = (POD2)$, $PG2 = (POD4)$ and to the address set containing the pods’ IPs $AS1 = (POD2-IP, POD4-IP)$. The ACL is (implicitly) applied to the node switches.

2. **ovn-northds** create SB relevant logical flows

3. **ovn-controllers** process SB updates and installs all required openflows
OVN-K8S traffic patterns (IC)

Node 1 (AZ 1)

- Northbound DB 1
- Southbound DB 1
- ovn-northd-1

- ovn-worker-1
- GR-worker-1
- join_switch-1
- ovn_cluster_router-1
- transit-switch

- POD1
- POD2
- OVS DB Node 1

Node 2 (AZ 2)

- Northbound DB 2
- Southbound DB 2
- ovn-northd-2

- ovn-worker-2
- GR-worker-2
- join_switch-2
- ovn_cluster_router-2
- transit-switch

- POD3
- POD4
- OVS DB Node 2

E-W Pods on node1 (AZ1):
10.244.0.0/24 via 169.254.0.1

N-S return traffic via LB on GR-worker1:
100.65.0.0/24 via 169.254.0.1

N-S return traffic via LB on GR-worker1:
(to bypass /24 src-policy route): 100.65.0.2/32 via 169.254.0.1

Default route:
0.0.0.0/0 via 100.66.0.2
1. IGMP snooping enabled on node switches
2. IGMP relay enabled on each AZ’s ovn_cluster_router
3. IGMP snooping enabled on the transit switch
4. IGMP reports forwarded by the cluster routers
5. IP multicast efficiently forwarded
ovnkube-master should create remote chassis in its zone Southbound database for nodes belonging to other zones.

ovnkube-master should (in its zone Northbound database)
- create transit switch
- transit switch ports for zone nodes and remote nodes
- connect ovn_cluster_router to transit_switch
- Add routes in the ovn_cluster_router for interconnection
- Centralized service running on the cluster master nodes
- Takes care of subnet allocation, unique id for each node, transit switch subnet allocation, egress ip node allocation etc.
- Doesn’t connect to OVN databases.

Subnet management (ovnkube-cluster-manager)
Preliminary Scale Testing
- 48 physical machines
  - 64 core Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
  - 187Gi RAM

- Kind kubernetes deployment with ovn-k8s CNI using ovn-kind-heater [1]

- 3 kind nodes (with master role) deployed on 1 physical machine.

- 188 kind worker nodes deployed across 47 physical machines.

https://github.com/numansiddique/kind/tree/join_support_v3
• Kubelet-density light test using kube-burner
  ○ Creates 250 pods per node. Total pods - 250 * 188 = 47000
  ○ Measures P99, P95, MAX and AVG time taken for the pods to be in Ready state.

• Memory and CPU utilization metrics using kube-prometheus.
ovn-k8s master deployment resources
- ovnkube-master deployment
  - Deployed on 3 master nodes
  - Containers
    - ovnkube-master
    - ovn-northd

- ovnkube-db deployment
  - Deployed on 3 master nodes
  - RAFT NB and SB cluster
  - Containers
    - NB ovsdb-server
    - SB ovsdb-server

- ovnkube-node daemonset
  - Deployed on all nodes (3 + 188)
  - Containers
    - ovnkube-node
    - ovn-controller

ovn-k8s interconnect deployment resources
- ovnkube-local daemonset
  - Deployed on all nodes (3 + 188)
  - Containers
    - ovnkube-local-master
    - ovn-northd
    - NB ovsdb-server
    - SB ovsdb-server
    - ovnkube-node
    - ovn-controller
ovn-k8s upstream vs ovn-k8s interconnect - kube-burner results

The graph shows a comparison between ovn-k8s upstream and ovn-k8s ic for Containers Ready time (in seconds). The bars represent the performance at P50, P99, and Avg across the two versions.
ovn-k8s upstream deployment: ovnkube master pod usage

ovnkube-master pod

CPU ~ 2.2

ovnkube-master pod has

- ovnkube-master container
- ovn-northd container
- Runs only on master nodes (3 nodes)

<table>
<thead>
<tr>
<th></th>
<th>ovn-northd</th>
<th>ovnkube-master</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1.6</td>
<td>0.6</td>
</tr>
<tr>
<td>Mem (RSS)</td>
<td>824 MiB</td>
<td>1024 MiB</td>
</tr>
</tbody>
</table>
ovnkube-db pod has

- Northbound ovsdb-server
- Southbound ovsdb-server

<table>
<thead>
<tr>
<th></th>
<th>NB server</th>
<th>SB server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>0.12</td>
<td>0.2</td>
</tr>
<tr>
<td>Mem (RSS)</td>
<td>230 MiB</td>
<td>844 MiB</td>
</tr>
</tbody>
</table>
ovn-k8s upstream deployment: ovnkube node pod usage

- ovnkube-node
- ovn-controller

<table>
<thead>
<tr>
<th></th>
<th>ovnkube-node</th>
<th>ovn-controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>0.025</td>
<td>0.17</td>
</tr>
<tr>
<td>Mem (RSS)</td>
<td>40 MiB</td>
<td>560 MiB</td>
</tr>
</tbody>
</table>
ovnkube-local pod has containers
- ovnkube-local-master
- ovn-northd container
- NB ovsdb-server
- SB ovsdb-server
- ovnkube-node
- ovn-controller

Pod CPU usage ~ 2 cores
Pod Mem (RSS) usage ~ 800 MiB
ovn-k8s ic deployment: ovnkube local pod usage (in detail)

NB ovsdb-server CPU ~ 0.06

SB ovsdb-server CPU ~ 0.5
ovn-k8s ic deployment: ovnkube local pod usage (in detail)

ovn-northd CPU ~ 0.8

ovn-controller CPU ~ 0.25
ovnkube-local-master
- CPU ~ 0.6
- Mem (RSS) ~ 600 MiB
ovnkube-node
- CPU ~ 0.025
- Mem (RSS) ~ 44 MiB
(Recap)

**ovn-k8s master deployment**
- ovnkube-master, ovn-northd and DB servers runs only on master (3) nodes.
- ovnkube-node and ovn-controller runs on all nodes

**ovn-k8s IC deployment**
- All the services run on all nodes.
Service comparison

ovn-k8s master deployment

- ovnkube-master, ovn-northd and DB servers runs only on master (3) nodes.
- ovnkube-node and ovn-controller runs on all nodes

ovn-k8s IC deployment

- All the services run on all nodes.

(Recap)

ovn-k8s master deployment

- ovnkube-master, ovn-northd and DB servers runs only on master (3) nodes.
- ovnkube-node and ovn-controller runs on all nodes

ovn-k8s IC deployment

- All the services run on all nodes.
<table>
<thead>
<tr>
<th>Resource</th>
<th>ovn-k8s upstream</th>
<th>ovn-k8s ic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>~0.275</td>
<td>~2</td>
</tr>
<tr>
<td>Mem (RSS)</td>
<td>~600 MiB</td>
<td>~800 MiB</td>
</tr>
</tbody>
</table>
Conclusions
IC Cons

- Data duplication - some cluster wide OVN configuration will have to be duplicated in every per node database. Overall more data stored across the cluster.
- Slight increase of the worker node CPU and memory usage.
- Will require refactoring OVN-k8s debugging tools - ovnkube-trace will need to now work across multiple databases.
- It ties ovn-kubernetes to the switch per node topology.
- Decentralized architecture, simplifying the deployment (no DBs in RAFT)
- Improved e2e latency when bringing up PODs (~30% average and P99 latency reduction)
- Improved resource usage on the central nodes (RSS/CPU needed for ovn-northd/NB/SB)
- No effort needed when developing new OVN-k8s features, allowing “hybrid” deployments:
  - group multiple nodes in the same availability zone to share the worker resource increase hit
Questions?