K3S flat network in EVE

Also known as No-NAT or shared switch network instance

Currently we plan to deploy K3S using two Ethernet ports per edge node. One Internet-facing management port which also has the port-map to reach the Trafeik load balancer on one edge-node, plus an internal Ethernet port connecting all of the edge-nodes without any NAT. That ensures that the master(s) and workers can communicate with each other without any NAT.

However, we'd like to avoid the complexity of that extra Ethernet port and Ethernet switch to connect them together. This document explores avoiding that by being able to have the app instances on the edge-nodes be able to communicate without NAT on the same port which is used by EVE for management.

Deployment model

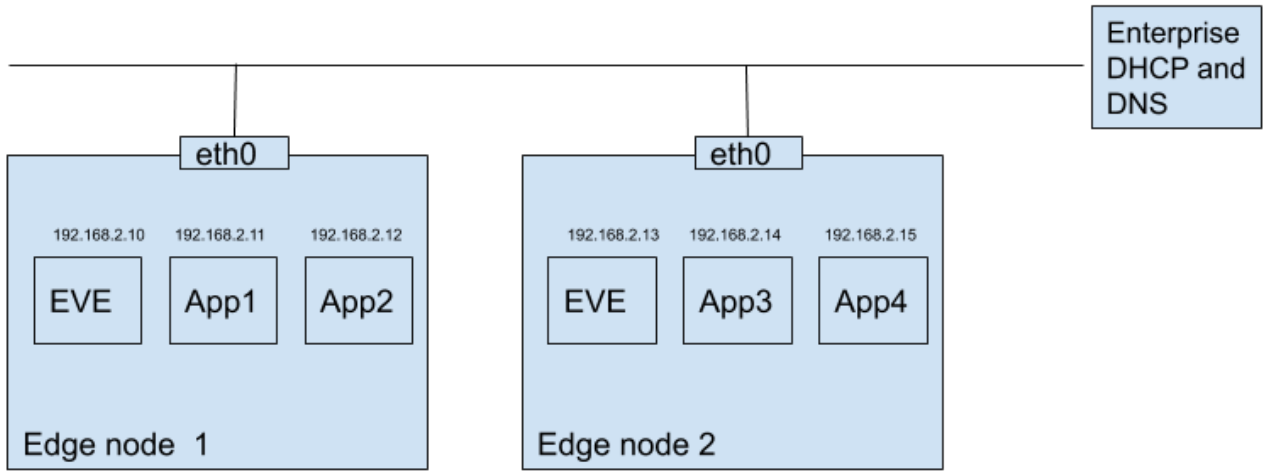

Figure 1: Deployment model

Conceptually this looks like a switch network instance which is attached to the management port. Note that conceptually that is easy, however our current implementation and use of Linux bridges etc prevents that. We'd like to be able to add some virtual interfaces to eth0 for the app instances and do so in a way where the app instances can communicate between themselves even if eth0 is not working. That prevents using macvlan. And we can't just attach a Linux bridge to eth0 since that breaks the use of eth0 for EVE's microservices. Hence the implementation becomes a bit complex as specified below.

We currently have checks in EVE and probably also in the UI and zedcloud to reject configurations combining "Management" and switch network instances. Once we have figured out the EVE side of things we need to look into disabling those checks.

New implementation

To enable the above we need to deploy a Linux bridge for each Ethernet port used by EVE. A minor detail is that we can not do that for LTE and WiFi ports since they are not capable of doing Ethernet bridging by having multiple Ethernet addresses etc.

In doing so we want to preserve the Ethernet address used by EVE. That is highly desirable so that when EVE is updated to the version with this support it will have the same MAC address as before the update, hence a DHCP server would give EVE the same IP address as before the update of EVE.

We also want to minimize the complexity of the implementation, which we do by making all of EVE believe it configures eth0 when in fact they configure a bridge vif which is attached to eth0. Thus the heavy lifting will be in the network interface manager to set up things differently and at the end of that there is an eth0 with an IP address used by EVE, but also a Linux bridge which other vifs can be attached to.

Note that today we have an issue with MAC address uniqueness when multiple edge-nodes use a switch network instance to connect to each other since we pick the MAC address for the vif based on the one byte application number so the first app instance on a given edge-node will have a fixed MAC address for their vifs; 00:16:3e:00:01:01. That currently needs to be overridden manually in the UI. In this new approach we will set them automatically to unique MACs based on the UUID of the app instance + interface number.

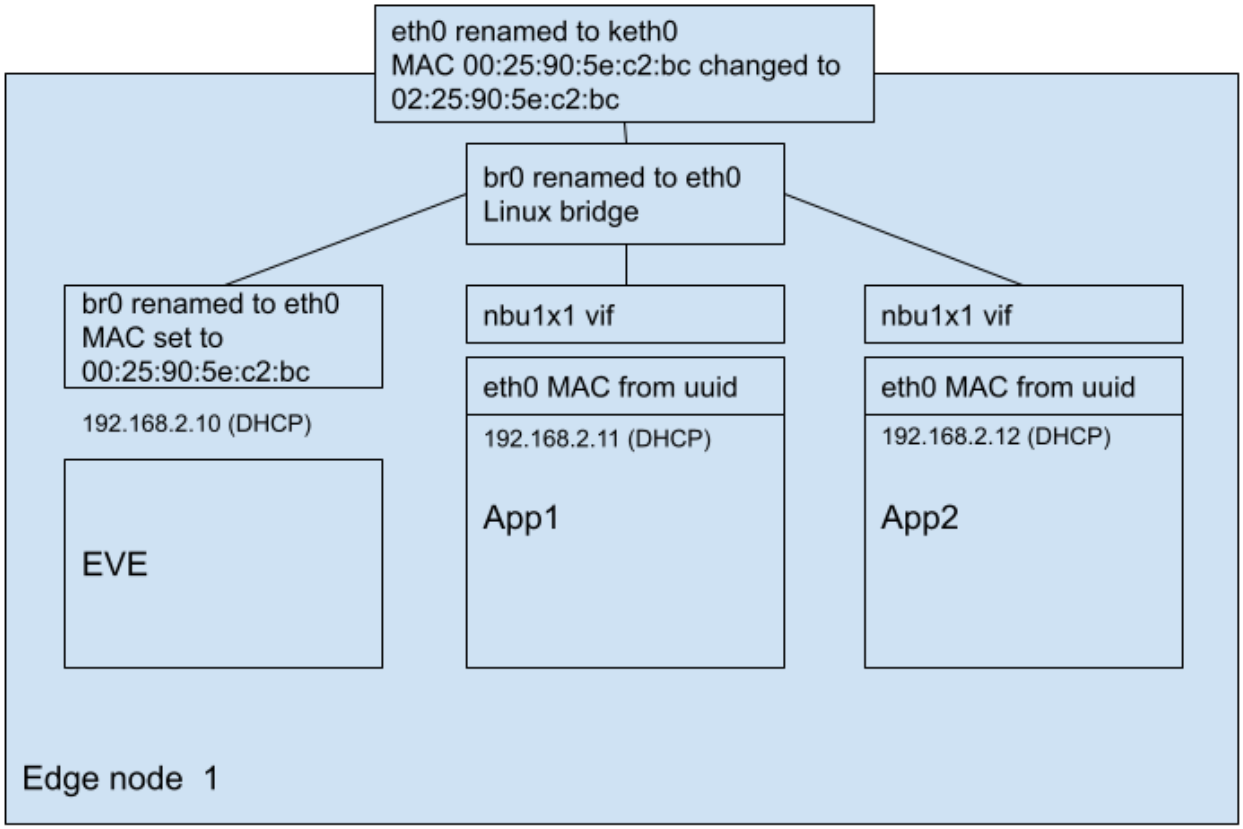

Figure 2: New implementation

The Linux bridge (br0 in the figure) is actually two logical components; a proper bridge where different Ethernet ports (physical or virtual) are attached to the bridge using brctl, and a designated IP interface (the head of the bridge) on which one can configure IP. Both of those have the same name but they are functionally separate hence shown as separate boxes in the figure. Typically they are not shown as separate items.

As shown above the br0 renamed to eth0 has the MAC address which was previously assiged to eth0 (renamed to keth0). But keth0 still needs to have a unique MAC address to avoid any issues. We ensure that by using the local bit (0x02) in the MAC address.

MAC addresses

App instances

Instead of using the fixed 00:16:3e:00:X:Y where X is a 0-255 local bridge number and Y is a 0-255 app number we will ensure network uniqueness by taking the App instance UUID plus the interface number for the app and hashing them down to 24 bits, and prefixing with 02:16:3e (with the the local bit set). If 2^^12 i.e., 4000 app instances are connected to the same Ethernet switch there will be a significant risk of collisions (about 50%) between a pair of them, but we expect the number of app instances connected to the same Ethernet switch to be order of 10 and not more, so this hashing should be OK.

EVE, physical ports, and bridges

Today EVE goes out with DHCP by default to get an IP address on the management ports. That relies on the MAC address assigned to the eth0 port. In the new implementation that will happen using the MAC address assigned to the designated bridge port br0. To make sure we present the same MAC address to the DHCP server we will set the local bit in the MAC address on the physical port and reuse the original MAC address from the physical port on br0.

Supporting local and cloud network instances etc

Other network instances have zero or more uplink ports which could refer to eth0 and/or eth1. Those will see no change, including when there is a switchover due to probing, since the eth0 and eth1 IP interfaces still exist under the same name.

Handle XEN renaming PCI interfaces

EVE has code to handle the case when some Ethernet PCI controller is assigned to pci-back, and then later when it is assigned back to dom0 the Ethernet ports are numbered differently. Thus was what called eth4 at initial boot can come back as eth0. EVE detects this using types.PciLongToIfname() and that can get confused by the proposed “eth0” vs. “keth0” renaming above. Also, domainmgr looks at the MAC address to check that it didn’t change and it can be confused by the setting of the local bit in the MAC address for eth0. Perhaps mask out the local bit in getMacAddr() to avoid any confusion.

Reference - old/current implementation

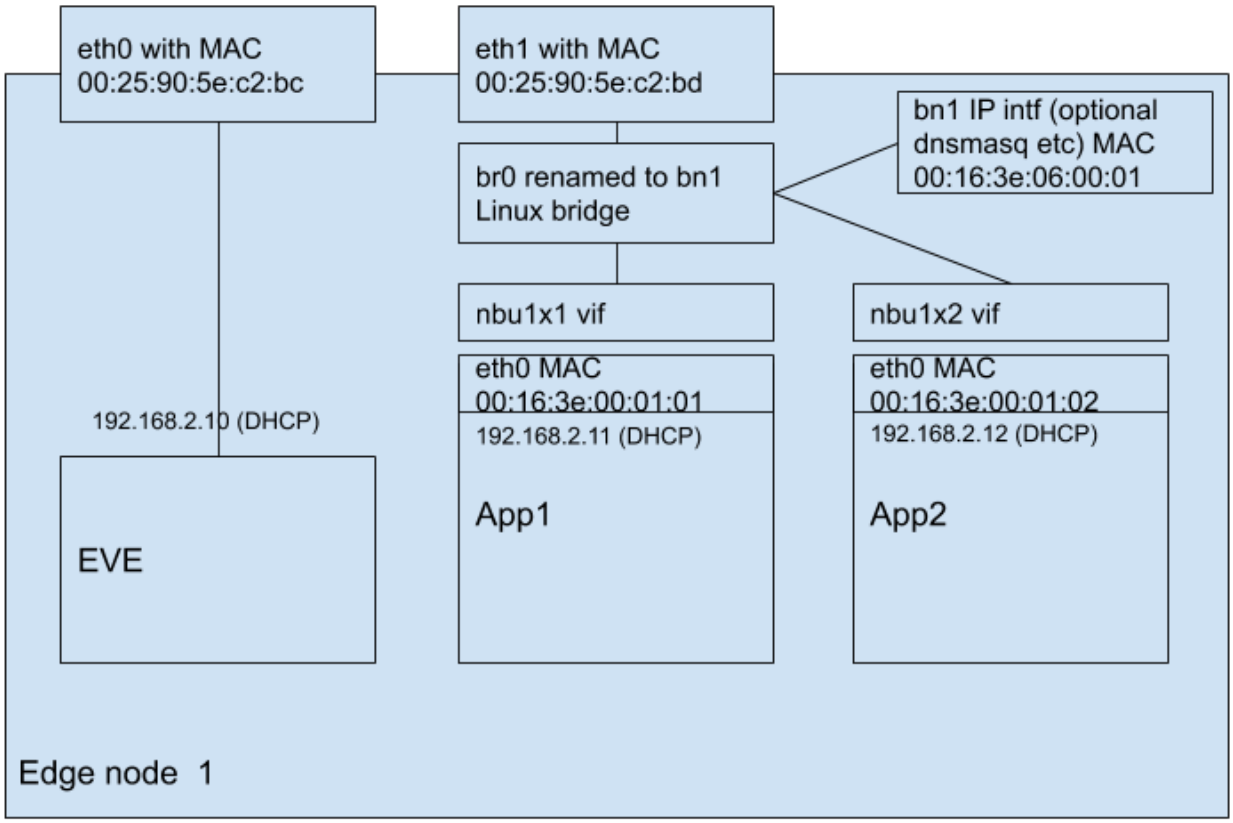

Figure 3: Old implementation