Connecting a Multi-Regional Kubernetes Cluster with Vita on AWS EC2

Interstellar Ventures

Monday, 16 March 2020

Table of Contents

In this demo we will setup an intercontinental IPv4 VPN between two data centers using *Vita* (https://github.com/inters/vita), and run a distributed Kubernetes cluster across both regions. Network traffic between the two regions will be routed via a Vita tunnel.

1 Lab Setup

AWS EC2 was chosen for this demo for its low barrier of access, and having flexible networking options. We run NixOS (*19.09.981.205691b7cbe x86_64-linux* (https://console.aws.amazon.com/ec2/home?region=eu-west-3#launchAmi=ami-09aa175c7588734f7)) on all instances to ensure the setup is reproducible.

We configure a VPC for each region with distinct private subnets. We will call them *vpc-paris* and *vpc-tokyo*.

VPC	Subnet	Default Gateway
vpc-paris	172.31.0.0/16	172.31.0.1
vpc-tokyo	172.32.0.0/16	172.32.0.1

Region VPCs.

In each region we create two EC2 instances. One instance `c5.xlarge` to host a Vita node, and one `c5.large` instance to host a Kubernetes node.

Instance	**VPC**	**Type**
vita-paris	*vpc-paris*	`c5.xlarge`
vita-tokyo	*vpc-toyko*	`c5.xlarge`
kube-node-paris	*vpc-paris*	`c5.large`
kube-node-tokyo	*vpc-tokyo*	`c5.large`

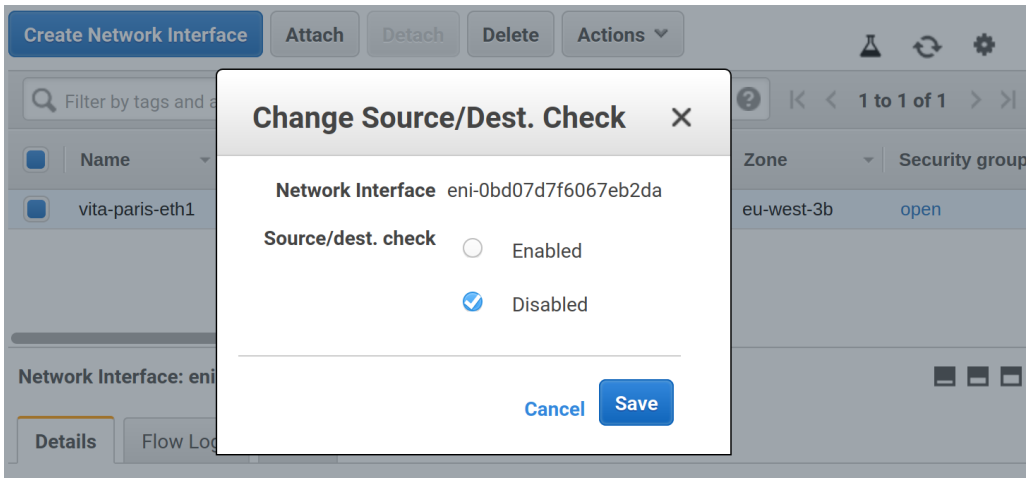
EC2 Instances.

The Vita nodes are each assigned four elastic network interfaces (ENA). The first interface is used as a management interface to SSH into the instance, the second interface will be the private interface used by Vita, and the remaining interfaces will be the public interfaces used by Vita (one for each queue—we will assign each Vita instance two CPU cores).

Instance	**Interface**	**Private IP**	**Public IP**
vita-paris	eth0	172.31.1.10	203.0.113.1
vita-paris	eth1	172.31.1.20	None
vita-paris	eth2	172.31.1.30	203.0.113.2
vita-paris	eth3	172.31.1.40	203.0.113.3
vita-tokyo	eth0	172.32.1.10	203.0.113.4
vita-tokyo	eth1	172.32.1.20	None
vita-tokyo	eth2	172.32.1.30	203.0.113.5
vita-tokyo	eth3	172.32.1.40	203.0.113.6

Vita node ENA configuration.

Important! The private interface (eth1) for each Vita instance must have its "Source/dest. check" option disabled in order for it to be able to forward packets.



Disabling "Source/dest. check" for an interface. (*Network Interfaces* > *Action* > *Change Source/dest. check*)

The Kubernetes nodes are each assigned a single interface. Note that the public IPs here are only used for management (i.e., to SSH into the instances).

Instance	**Interface**	**Private IP**	**Public IP**
kube-node-paris	eth0	172.31.1.50	203.0.113.7
kube-node-tokyo	eth0	172.32.1.50	203.0.113.8

K8s node ENA configuration.

Note. For convenience during testing we assign a permissive security group to all network interfaces which allows all incoming traffic. In a real setup, for Vita nodes, you would only allow ICMP and SSH on the management interfaces, ICMP, ESP and UDP/303 (for the AKE protocol) on the public interface, and restrict the private interface as needed.

2 Configuring Vita

We will use XDP to drive the EC2 instances' ENA virtual NICs. For that we need a Linux kernel with `XDP_SOCKETS` support, and a recent ENA driver that support `ethtool -set-channels`.

```
boot.kernelPackages = let
  linux_pkg = { fetchurl, buildLinux, ... } @ args:

    buildLinux (args // rec {
      version = "5.5.0";
      modDirVersion = version;

      src = fetchurl {
        url = "https://github.com/torvalds/linux/archive/v5.5.tar.gz";
        sha256 = "87c2ecdd31fcf479304e95977c3600b3381df32c58a260c72921a6bb7ea629
      };

      extraConfig = ''
        XDP_SOCKETS y
      '';

      extraMeta.branch = "5.5";
    } // (args.argsOverride or {}));
  linux = pkgs.callPackage linux_pkg{};
in
  pkgs.recurseIntoAttrs (pkgs.linuxPackagesFor linux);
```

NixOS configuration for Vita instances: use a kernel with recent ENA driver and `XDP_SOCKETS` enabled.

The ENA driver currently do not support `ethtool -config-nfc` beyond modifying the RSS hash, so we will use a separate ENA with a single combined queue for each public interface. We isolate CPU cores 2 and 3 of the `c5.xlarge` instances for use by Vita.

```
boot.kernelParams = [ "isolcpus=2-3" ]
boot.postBootCommands = ''
  ethtool --set-channels eth1 combined 2
  ethtool --set-channels eth2 combined 1
  ethtool --set-channels eth3 combined 1
'';
```

NixOS configuration to isolate CPU cores 2-3, and set the desired number of queues on our network interfaces.

Finally, we can clone and install Vita on the instance via `nix-env`, and run it on the isolated CPU cores.

```
git clone https://github.com/inters/vita.git
nix-env -i -f vita
```

Clone and install Vita via `nix-env`.

```
vita --name paris --xdp --cpu 2,3
```

Run Vita in XDP mode, with its worker threads bound to CPU cores 2-3.

We configure the Vita nodes as follows:

- Configure eth1 as the IPv4 private interface, use 172.31.0.1 as next hop (we let the default router forward packets to the subnet's hosts). The private interface will process packets on two queues in RSS mode.
- Configure eth2 and eth3 as IPv4 public interfaces for queue 1 and 2. Both use first and only device queue of their interface by setting `device-queue` to 1. Further, we specify the NATed public IP via the `nat-ip` option. Again, we set the default gateway as the next hop.
- Set the MTU to 1440 to account for encapsulation overhead as the EC2 ENA interfaces provide us with a standard MTU of 1500 by default.
- Configure an IPv4 route to the remote Vita node. Besides the remote subnet, the route is assigned two gateways identified by the public IP address of the remote public interface (one per queue).

```
private-interface4 {
  ifname eth1;
  mac 0a:cc:b1:e4:24:d2;
  ip 172.31.1.20;
  nexthop-ip 172.31.0.1;
}

public-interface4 {
  queue 1;
  ifname eth2;
  device-queue 1;
  mac 0a:42:13:9e:a8:ae;
  ip 172.31.1.30;
  nat-ip 203.0.113.2;
  nexthop-ip 172.31.0.1;
}
public-interface4 {
  queue 2;
  ifname eth3;
  device-queue 1;
  mac 0a:33:39:04:9c:ac;
  ip 172.31.1.40;
  nat-ip 203.0.113.3;
  nexthop-ip 172.31.0.1;
}

mtu 1440;

route4 {
  id tokyo;
  gateway { queue 1; ip 203.0.113.5; }
  gateway { queue 2; ip 203.0.113.6; }
  net "172.32.0.0/16";
  preshared-key "ACAB129A...";
  spi 1234;
}
```

Vita configuration for *vita-paris*.

To apply the configuration we can use `snabb config`. To see what the Vita node is doing we can use `snabb config get-state` to query its run-time statistics, and `snabb top` to watch its internal links in real-time.

```
snabb config set paris / < vita-paris.conf
```

Apply Vita configuration via `snabb config`.

```
snabb config get-state paris /gateway-state
```

Use `snabb config get-state` to query run-time statistics.

3 Kubernetes Cluster Setup

We configure *kube-node-paris* as a Kubernetes master node. For simplicity of illustration we add an extra static host *kube-node-tokyo* using its private IP in the remote subnet, and disable the firewall.

```
networking.hostName = "kube-node-paris";
networking.extraHosts = ''
  127.0.0.1 kube-node-paris
  172.32.1.50 kube-node-tokyo
'';
networking.firewall.enable = false;

services.kubernetes = {
  roles = ["master"];
  masterAddress = "kube-node-paris";
  apiserverAddress = "https://kube-node-paris:6443";
};
```

NixOS configuration for *kube-node-paris*.

We configure *kube-node-paris* to route packet destined to the remote subnet via *vita-paris*, and set the route's MTU accordingly. I.e., we forward packets for the subnet directly to the Vita node, bypassing the default gateway.

```
ip route add 172.32.0.0/16 via 172.31.1.20 dev eth0 mtu 1440
```

Route packets to tokyo subnet 172.32.0.0/16 via the private interface of *vita-paris*.

We also take note of the "apitoken" generated for the Kubernetes cluster.

```
/var/lib/kubernetes/secrets/apitoken.secret
```

Obtain the "apitoken" of the Kubernetes master node.

Kube-node-tokyo is configured as a regular Kubernetes node, with *kube-node-paris* as its master.

```
networking.hostName = "kube-node-tokyo";
networking.extraHosts = ''
  127.0.0.1 kube-node-tokyo
  172.31.1.50 kube-node-paris
'';
networking.firewall.enable = false;

services.kubernetes = {
  roles = ["node"];
  masterAddress = "kube-node-paris";
  apiserverAddress = "https://kube-node-paris:6443";
};
```

NixOS configuration for *kube-node-tokyo*.

Again we configure the routing table to route packets for the paris subnet through the Vita tunnel.

```
ip route add 172.31.0.0/16 via 172.32.1.20 dev eth0 mtu 1440
```

Route packets to paris subnet 172.31.0.0/16 via the private interface of *vita-tokyo*.

Finally, have the Kubernetes node join the cluster using the cluster's "apitoken".

```
nixos-kubernetes-node-join < apitoken.secret
```

4 Verifying the Setup

You should be able to verify the setup and test connectivity using `ping`, `traceroute`, and `iperf` on *kube-node-paris* and *kube-node-tokyo*. Further, you can verify that the cluster assembled successfully via `kubectl` on *kube-node-paris*.

```
export KUBECONFIG=/etc/kubernetes/cluster-admin.kubeconfig

# List cluster nodes
kubectl get nodes

# Pods running on kube-node-tokyo?
kubectl --namespace kube-system get pods -o wide

# Any error events?
kubectl --namespace kube-system get events
```