Cluster Networking

Kubernetes approaches networking somewhat differently than Docker does by default. There are 4 distinct networking problems to solve:

1. Highly-coupled container-to-container communications: this is solved by pods and `localhost` communications.
2. Pod-to-Pod communications: this is the primary focus of this document.
3. Pod-to-Service communications: this is covered by services.
4. External-to-Service communications: this is covered by services.

Summary

Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. Every pod gets its own IP address so you do not need to explicitly create links between pods and you almost never need to deal with mapping container ports to host ports. This creates a clean, backwards-compatible model where pods can be treated much like VMs or physical hosts from the perspectives of port allocation, naming, service discovery, load balancing, application configuration, and migration.

There are requirements imposed on how you set up your cluster networking to achieve this.

Docker model

Before discussing the Kubernetes approach to networking, it is worthwhile to review the “normal” way that networking works with Docker. By default, Docker uses host-private networking. It creates a virtual bridge, called `docker0` by default, and allocates a subnet from one of the private address blocks defined in RFC1918 for that bridge. For each container that Docker creates, it allocates a virtual Ethernet device (called `veth`) which is attached to the bridge. The veth is mapped to appear as `eth0` in the container, using Linux namespaces. The in-container `eth0` interface is given an IP address from the bridge’s address range.

The result is that Docker containers can talk to other containers only if they are on the same machine (and thus the same virtual bridge). Containers on different machines can not reach each other - in fact they may end up with the exact same network ranges and IP addresses.

In order for Docker containers to communicate across nodes, there must be allocated ports on the machine’s own IP address, which are then forwarded or proxied to the containers. This obviously means that containers must either coordinate which ports they use very carefully or ports must be allocated dynamically.

Kubernetes model

Coordinating ports across multiple developers is very difficult to do at scale and exposes users to cluster-level issues outside of their control. Dynamic port allocation brings a lot of complications to the system - every application has to take ports as flags, the API servers have to know how to insert dynamic port numbers into configuration blocks, services have to know how to find each other, etc. Rather than deal with this, Kubernetes takes a different approach.

Kubernetes imposes the following fundamental requirements on any networking implementation (barring any intentional network segmentation policies):

- all containers can communicate with all other containers without NAT
- all nodes can communicate with all containers (and vice-versa) without NAT
- the IP that a container sees itself as is the same IP that others see it as

What this means in practice is that you can not just take two computers running Docker and expect Kubernetes to work. You must ensure that the fundamental requirements are met.

This model is not only less complex overall, but it is principally compatible with the desire for Kubernetes to enable low-friction porting of apps from VMs to containers. If your job previously ran in a VM, your VM had an IP and could talk to other VMs in your project. This is the same basic model.

Until now this document has talked about containers. In reality, Kubernetes applies IP addresses at the `Pod` scope - containers within a `Pod` share their network namespaces - including their IP address. This means that containers within a `Pod` can all reach each other's ports on `localhost`. This does imply that containers within a `Pod` must coordinate port usage, but this is no different than processes in a VM. This is called the "IP-per-pod" model. This is implemented, using Docker, as a "pod container" which holds the network namespace open while "app containers" (the things the user specified) join that namespace with Docker's `--net=container:<id>` function.

As with Docker, it is possible to request host ports, but this is reduced to a very niche operation. In this case a port will be allocated on the host `Node` and traffic will be forwarded to the `Pod`. The `Pod` itself is blind to the existence or non-existence of host ports.

How to achieve this

There are a number of ways that this network model can be implemented. This document is not an exhaustive study of the various methods, but hopefully serves as an introduction to various technologies and serves as a jumping-off point.

The following networking options are sorted alphabetically - the order does not imply any preferential status.

ACI

Cisco Application Centric Infrastructure offers an integrated overlay and underlay SDN solution that supports containers, virtual machines, and bare metal servers. ACI provides container networking integration for ACI. An overview of the integration is provided here.

Big Cloud Fabric from Big Switch Networks

Big Cloud Fabric is a cloud native networking architecture, designed to run Kubernetes in private cloud/on-premise environments. Using unified physical & virtual SDN, Big Cloud Fabric tackles inherent container networking problems such as load balancing, visibility, troubleshooting, security policies & container traffic monitoring.

With the help of the Big Cloud Fabric's virtual pod multi-tenant architecture, container orchestration systems such as Kubernetes, RedHat Openshift, Mesosphere DC/OS & Docker Swarm will be natively integrated along side with VM orchestration systems such as VMware, OpenStack & Nutanix. Customers will be able to securely inter-connect any number of these clusters and enable inter-tenant communication between them if needed.

BCF was recognized by Gartner as a visionary in the latest Magic Quadrant. One of the BCF Kubernetes on premise deployments (which includes Kubernetes, DC/OS & VMware running on multiple DCs across different geographic regions) is also referenced here.

Cilium

Cilium is open source software for providing and transparently securing network connectivity between application containers. Cilium is L7/HTTP aware and can enforce network policies on L3-L7 using an identity based security model that is decoupled from network addressing.

Contiv

Contiv provides configurable networking (native l3 using BGP, overlay using vxlan, classic l2, or Cisco-SDN/ACI) for various use cases. Contiv is all open sourced.

Contrail

Contrail, based on OpenContrail, is a truly open, multi-cloud network virtualization and policy management platform. Contrail / OpenContrail is integrated with various orchestration systems such as Kubernetes, OpenShift, OpenStack and Mesos, and provides different isolation modes for virtual machines, containers/pods and bare metal workloads.

Flannel

Flannel is a very simple overlay network that satisfies the Kubernetes requirements. Many people have reported success with Flannel and Kubernetes.

Google Compute Engine (GCE)

For the Google Compute Engine cluster configuration scripts, advanced routing is used to assign each VM a subnet (default is `/24` - 254 IPs). Any traffic bound for that subnet will be routed directly to the VM by the GCE network fabric. This is in addition to the “main” IP address assigned to the VM, which is NAT'ed for outbound internet access. A linux bridge (called `cbr0`) is configured to exist on that subnet, and is passed to docker's `--bridge` flag.

Docker is started with:

```
DOCKER_OPTS="--bridge=cbr0 --iptables=false --ip-masq=false"
```

This bridge is created by Kubelet (controlled by the `--network-plugin=kubenet` flag) according to the `Node`'s `spec.podCIDR`.

Docker will now allocate IPs from the `cbr-cidr` block. Containers can reach each other and `Nodes` over the `cbr0` bridge. Those IPs are all routable within the GCE project network.

GCE itself does not know anything about these IPs, though, so it will not NAT them for outbound internet traffic. To achieve that an iptables rule is used to masquerade (aka SNAT - to make it seem as if packets came from the `Node` itself) traffic that is bound for IPs outside the GCE project network (10.0.0.0/8).

```
iptables -t nat -A POSTROUTING ! -d 10.0.0.0/8 -o eth0 -j MASQUERADE
```

Lastly IP forwarding is enabled in the kernel (so the kernel will process packets for bridged containers):

```
sysctl net.ipv4.ip_forward=1
```

The result of all this is that all `Pods` can reach each other and can egress traffic to the internet.

Kube-router

Kube-router is a purpose-built networking solution for Kubernetes that aims to provide high performance and operational simplicity. Kube-router provides a Linux LVS/IPVS-based service proxy, a Linux kernel forwarding-based pod-to-pod networking solution with no overlays, and iptables/ipset-based network policy enforcer.

L2 networks and linux bridging

If you have a “dumb” L2 network, such as a simple switch in a “bare-metal” environment, you should be able to do something similar to the above GCE setup. Note that these instructions have only been tried very casually - it seems to work, but has not been thoroughly tested. If you use this technique and perfect the process, please let us know.

Follow the “With Linux Bridge devices” section of this very nice tutorial from Lars Kellogg-Stedman.

Multus (a Multi Network plugin)

Multus is a Multi CNI plugin to support the Multi Networking feature in Kubernetes using CRD based network objects in Kubernetes.

Multus supports all reference plugins (eg. Flannel, DHCP, Macvlan) that implement the CNI specification and 3rd party plugins (eg. Calico, Weave, Cilium, Contiv). In addition to it, Multus supports SRIOV, DPDK, OVS-DPDK & VPP workloads in Kubernetes with both cloud native and NFV based applications in Kubernetes.

NSX-T

VMware NSX-T is a network virtualization and security platform. NSX-T can provide network virtualization for a multi-cloud and multi-hypervisor environment and is focused on emerging application frameworks and architectures that have heterogeneous endpoints and technology stacks. In addition to vSphere hypervisors, these environments include other hypervisors such as KVM, containers, and bare metal.

NSX-T Container Plug-in (NCP) provides integration between NSX-T and container orchestrators such as Kubernetes, as well as integration between NSX-T and container-based CaaS/PaaS platforms such as Pivotal Container Service (PKS) and Openshift.

Nuage Networks VCS (Virtualized Cloud Services)

Nuage provides a highly scalable policy-based Software-Defined Networking (SDN) platform. Nuage uses the open source Open vSwitch for the data plane along with a feature rich SDN Controller built on open standards.

The Nuage platform uses overlays to provide seamless policy-based networking between Kubernetes Pods and non-Kubernetes environments (VMs and bare metal servers). Nuage’s policy abstraction model is designed with applications in mind and makes it easy to declare fine-grained policies for applications.The platform’s real-time analytics engine enables visibility and security monitoring for Kubernetes applications.

OpenVSwitch

OpenVSwitch is a somewhat more mature but also complicated way to build an overlay network. This is endorsed by several of the “Big Shops” for networking.

OVN (Open Virtual Networking)

OVN is an opensource network virtualization solution developed by the Open vSwitch community. It lets one create logical switches, logical routers, stateful ACLs, load-balancers etc to build different virtual networking topologies. The project has a specific Kubernetes plugin and documentation at ovn-kubernetes.

Project Calico

Project Calico is an open source container networking provider and network policy engine.

Calico provides a highly scalable networking and network policy solution for connecting Kubernetes pods based on the same IP networking principles as the internet. Calico can be deployed without encapsulation or overlays to provide high-performance, high-scale data center networking. Calico also provides fine-grained, intent based network security policy for Kubernetes pods via its distributed firewall.

Calico can also be run in policy enforcement mode in conjunction with other networking solutions such as Flannel, aka canal, or native GCE networking.

Romana

Romana is an open source network and security automation solution that lets you deploy Kubernetes without an overlay network. Romana supports Kubernetes Network Policy to provide isolation across network namespaces.

Weave Net from Weaveworks

Weave Net is a resilient and simple to use network for Kubernetes and its hosted applications. Weave Net runs as a CNI plug-in or stand-alone. In either version, it doesn't require any configuration or extra code to run, and in both cases, the network provides one IP address per pod - as is standard for Kubernetes.

CNI-Genie from Huawei

CNI-Genie is a CNI plugin that enables Kubernetes to simultaneously have access to different implementations of the Kubernetes network model in runtime. This includes any implementation that runs as a CNI plugin, such as Flannel, Calico, Romana, Weave-net.

CNI-Genie also supports assigning multiple IP addresses to a pod, each from a different CNI plugin.

Other reading

The early design of the networking model and its rationale, and some future plans are described in more detail in the networking design document.

Create an Issue Edit this Page