Kubernetes Network Performance Measurement via Kubemark

Motivation

Currently, we have the ability to deploy Pods in a Kubernetes cluster and run data path performance measurement tests using various tools such as iperf and get performance metrics of the data-plane of the underlying CNI (Container Network Interface) provider in use. However, metrics gathered in such a manner are usually representative of ideal conditions rather than real-world situations. While runtimes such as containerd gather limited metrics such as CNI ADD latency, measuring such latency at large scale requires deployment of a large cluster and that is not cost-effective.

At this time, we are unaware of a facility that allows measuring CNI control plane performance metrics such as:

- 'Network-ready' latency: How long does it take for the CNI provider to allocate IP which is reachable by other pods/services as well as can reach other pods and services (internal and external)?
- Pod flux: How well does a CNI provider work under high pod load with pods rapidly arriving and departing nodes?
- Realistic cluster traffic model: How well does the CNI provider handle real-world network traffic pattern models in a cluster?

Kubemark is a performance testing tool for Kubernetes that allows simulating or modeling real-world clusters and allows to run experiments on them and gather metrics that provide insights into the performance of various K8s control components such as apiserver, controller, schedulers etc.

Kubemark does this cost-effectively by using pods that pretend to be nodes (hollow nodes) and fake kubelet that pretends to start pods (hollow kubelet) but gathers metrics at each eventful step in the process of running a real pod.

Goals & Non-goals

The broad goal of this project is to extend the Kubemark tool to allow for performance measurement of the K8s pod networking solution (CNI provider) at large scale in simulated K8s clusters.

Specifically, this effort intends to enable the following at large scale:

1. Pod ‘network-ready’ latencies.
2. Pod ‘network-ready’ throughput.
3. Basic, light-weight network traffic modeling.
4. Model application deployment & testing in K8s cluster.

The following non-goals (at least for the first version):

1. Act as a full-fledged network simulator or support network topology emulation.
2. Large scale performance of CoreDNS or its plugins (Needs more investigation to understand what metrics can be collected - deferred for now)
3. kube-proxy large scale performance (Not a direct goal as CNI implementations may use other LB mechanisms instead of iptables/ipvs that bypass/disable kube-proxy)
4. Network policy large scale performance (Needs more investigation to understand what metrics might make sense and the complexity involved)

Note: While Kubernetes may be able to leverage this to detect, in a limited way, regressions in scaling or performance from build-to-build or release-to-release, the primary goal in mind of this effort is to have a standard way to determine how CNI provider control plane performs in Kubernetes environment at scale and help detect regressions or drive improvements in the CNI drivers in a cost-effective manner using kubemark.

Design

This section details the following:

1. Modifications to Kubemark to enable deployment of real pods in a kubemark (not-so)-hollow-node.
2. Modifications to container runtime (containerd) to enable enhanced metrics collection
3. Modifications to container runtime (containerd) to enable light-weight traffic modeling.

Kubemark Changes

In order to deploy real pods, we need to use real kubelet. This is achieved as follows:

- Add new Dockerfile that contains kubelet from the build
    - It also contains the modified containerd discussed in the next section
    - Additionally, it contains various network perf testing tools such as ping, curl, conntrack, tcpdump, and iperf.
    - Include scripts to start real kubelet and containerd in the (not-so)-hollow-node.
- Add kubemark template file with ports and volume mounts needed to support the operation of real kubelet and the modified containerd.
- Modify kubemark cluster deployment scripts to support the option to deploy network performance testing kubemark cluster in addition to the current perf testing cluster.

Containerd Changes For Network Perf Metrics

Kubelet delegates container operations to the container runtime (containerd by default) via the CRI (Container Runtime Interface). containerd is responsible for pulling the container images from the repository, creating the pod sandbox, calling the CNI to set up the pod networking, creating and mounting the required volumes, creating the cgroups, and starting the containers. All these operations constitute the starting up of a pod.

In order to focus on measurement of network-ready latencies, we skip or mock all the other pod startup steps except for CNI_COMMAND ADD which invokes the configured CNI provider to set up the pod networking.

One the CNI provider returns with a successful response, we record / observe the total time it took from the very first ADD command to the point when the first ADD succeeded, in addition to the currently measured latency of the specific successful ADD command. This gives us the true latency of the CNI provider pod network setup process.

Once we have a successful CNI ADD and get an IP, we run basic tests using that IP (switching to pod IP netns) to control entities such as:

- Ping a control pod (real pod) on the same node as the current CNI ADD.
- Ping control pods on a random set of 5 other not-so-hollow-nodes.
- Ping a control service.
- Run multiple curl tests on a control service.

- Ping a service external to the cluster.
- Run curl test to a service external to the cluster.

Once these tests are successful, we have the true ‘network-ready’ latency that we can observe and record.

Network Performance Target Pod API

We use the pod labels as the API for kubemark to specify whether the pod in question is subject to above described network performance mocking and metrics gathering.

If the pod contains the label “netperf: true” then pod is subject to network simulation and performance measurement path in containerd.If this label is missing or not set to true, then the pod gets the normal containerd handling and real workload will be started.

This enables writing kubemark network performance tests that allow deployment of pods subject to network performance metrics at large scale.

Containerd Changes To Enable Lightweight Traffic Modeling

In addition to measuring control plane network-ready latencies as described above, this framework can be used to perform data plane performance measurements of newly connected ‘pods’ by running basic iperf tests from the pod’s IP against control pods which are hosting iperf servers.

The exact details of this is TBD, but the high-level idea is that network perf pod’s default container command can be leveraged for tests describing the target control entities and the nature of the test traffic to use by specifying the iperf command to run. Kubemark tests can direct containerd to run the specified commands upon pod ‘network ready’.

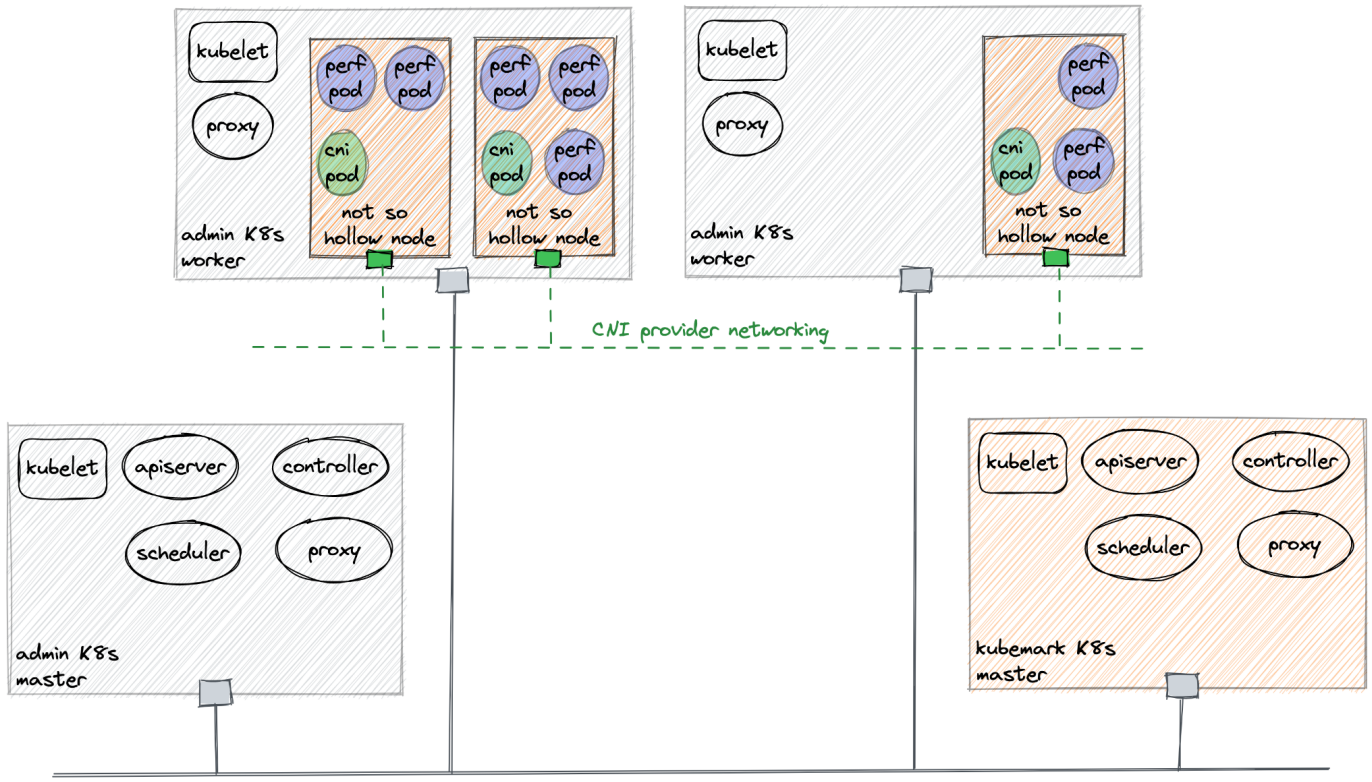
This will enable kubemark as a cost-effective, light-weight tool to perform K8s cluster network simulations.

Alternatives Considered

Modified Kubelet: Another alternative design considered was to extend the kubelet to directly invoke the CNI and record latency measurements. While this avoids the CRI roundtrip, it does create special code that needs to be maintained as kubelet evolves. For the time being, relying on contract assurances of CRI API calls and invoke mocks based on the label looks simpler and more maintainable.

Design Illustration

This section illustrates and describes the design with a simple depiction of a kubemark cluster with the changes described in the previous section.



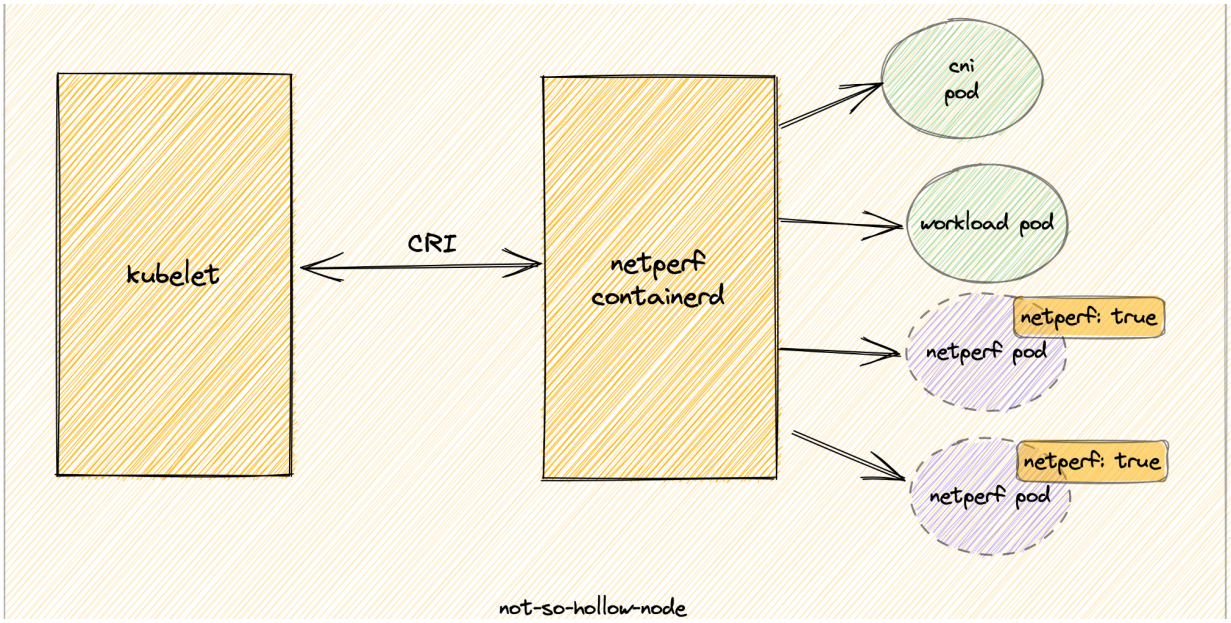
The above illustration shows a standard kubemark deployment consisting of an admin cluster with one master and two workers shown as nodes with gray background. This is a normal kubernetes cluster with K8s components such as apiserver, controller manager, scheduler, kubelet, and kube-proxy running on the master nodes while worker nodes run kubelet and kube-proxy processes. In GCE deployments, each of these nodes are VMs.

The kubemark cluster nodes are shown with an orange background. The kubemark master node is similar to the admin master node and is a standalone VM. And just like kubemark, the deployment creates pods that run on admin worker nodes and join the kubemark master as nodes.

The difference is that these kubemark nodes called not-so-hollow-nodes have the ability to run real workload pods, unlike the usual kubemark hollow-nodes which download the image and pretend to run the pods but don't actually run anything. The not-so-hollow-nodes are built to start the test-target CNI provider pods.

Containerd Design Illustration

The below illustration depicts how containerd handles pod sandbox and container run requests to provide the ability to measure network performance of the CNI providers at scale.



The full-fledged kubelet that replaces the hollow-kubelet in the original kubemark uses the runtime (netperf-containerd) via CRI to create pod sandbox and run the pod workloads.

- All regular pods will go through the normal sandbox creation path and will result in a running workload. These pods are shown above in green backfill.
- All pods labeled with ‘netperf: true’ will cause netperf-containerd to skip all pod sandbox and container creation steps and instead trigger a go routine that is tasked with pod network setup, network-readiness verification, and user-defined network model tests. These pods are shown above in blue backfill.

Test Plan

TBD

Risks and Concerns

1. Cost Effectiveness: This approach makes use of real kubelet and real containerd. This may limit the density or the number of kubemark worker nodes (pods) that may be run on an admin worker node, and may require admin worker nodes of higher CPU and memory configuration. While this will be less cost-effective than truly hollow-nodes in current kubemark design, it will very likely provide better cost-effectiveness than using real clusters. Just how cost-effective it really will be is yet to be seen. It is possible optimizing the kubelet pod-workers for this case may quickly yield significant benefits.