Installing and testing Kubernetes with Knative

Bill Brouwer 01/21

Introduction

The serverless paradigm has been around at least as long as AWS Lambda, launched somewhere on or before 2014. Of course, there are servers, network and objects like load balancers somewhere, however, in this computational approach virtually all the details are abstracted away. In this article I examine installing Kubernetes (K8s) and Knative and provide some results when running a highly task parallel, containerized workload on a commercial example of this serverless technology, Google Cloud Run.

Kubernetes

For this test, I scavenged some antique machines from the closet and began by installing Ubuntu 20.04, docker-ce and K8s on bare metal. At the time of writing, there doesn't appear to be a complete K8s repos for this release of Ubuntu, so I used the official repos for Xenial from Google, without problems. I encourage a quick search in order to find the most up to date repos for your Linux release. If I share too many links then I risk dating this article even more rapidly 😊

Here are the details of my initial configuration, and note that hostnames have been set to master or worker and make sure swap has been disabled on all nodes:

```
//network  1
1192.168.0.73 (master)
192.168.0.56 (worker01)
192.168.0.70 (worker02)

//network 2
192.168.1.101 (master)
192.168.1.102 (worker01)
192.168.1.100 (worker02)

//machine - worker 01 / 02
$ more / proc / cpuinfo | grep model\ name | head – 1
```

```
model name : Intel(R) Celeron(R) CPU  J1900  @ 1.99GHz

//machine – master
$ more / proc / cpuinfo | grep model\ name | head – 1
model name : Intel(R) Core(TM) i7 - 2600 CPU @ 3.40GHz

$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2",
GitCommit:"faecb196815e248d3ecfb03c680a4507229c2a56", GitTreeState:"clean",
BuildDate:"2021-01-13T13:25:59Z", GoVersion:"go1.15.5", Compiler:"gc",
Platform:"linux/amd64"}

$ docker --version
Docker version 20.10.2, build 2291f61
```

With this state or similar, you are ready to instantiate your cluster on master ala:

```
$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

which should elicit this response albeit with a different sha:

```
To start using your cluster, you need to run the following as a regular user:

        mkdir -p $HOME/.kube
        sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
        sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

        export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as
root:

kubeadm join 192.168.0.73:6443 --token mvhmrn.4xbogjjcof4ce8xm\
--discovery-token-ca-cert-hash
sha256:02d6df91202487f08735b9bfd8f963390b93a8e01fc9d031d86092629a233716
```

Follow these steps carefully, making sure that once the config has been set, that the pod network is indeed deployed before attempting to add nodes, eg., here flannel is used:

```
$ sudo KUBECONFIG=$HOME/.kube/config kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

And thereafter the join statement mentioned in the response earlier can be used on all the worker nodes. **Note that sudo is required for these latter steps.** As above, you may find the need to point explicitly to the KUBECONFIG.

Before proceeding to Knative installation, do make sure all pods are in a running state eg:

```
$ kubectl get pods -n kubesystem

NAME                                   READY   STATUS    RESTARTS   AGE
coredns-74ff55c5b-dndhh                1/1     Running   0          19h
coredns-74ff55c5b-zgc5r                1/1     Running   0          19h
etcd-master-node                       1/1     Running   0          19h
kube-apiserver-master-node             1/1     Running   7          19h
kube-controller-manager-master-node    1/1     Running   1          19h
kube-flannel-ds-9w54x                  1/1     Running   0          19h
kube-flannel-ds-w4zh7                  1/1     Running   0          19h
kube-flannel-ds-ww67n                  1/1     Running   0          19h
kube-proxy-6fjzm                       1/1     Running   0          19h
kube-proxy-mn42w                       1/1     Running   0          19h
kube-proxy-nwxmh                       1/1     Running   0          19h
kube-scheduler-master-node             1/1     Running   1          19h
```

Knative

The official website is (not surprisingly) the best source of up to date information; here we simply focus on the serving installation : https://knative.dev/docs/install/any-kubernetes-cluster/#installing-the-serving-component

The following installations are exclusively on the master node. Begin by installing Custom Resource Definitions and core components:

```
$ kubectl apply --filename
https://github.com/knative/serving/releases/download/v0.20.0/serving-crds.yaml

$ kubectl apply --filename
https://github.com/knative/serving/releases/download/v0.20.0/serving-core.yaml
```

For the networking layer, istio appears to be the most popular choice when browsing online blogs detailing experience with Knative. Install this component next, starting with the CLI, istioctl:

```
$ curl -sL https://istio.io/downloadIstioctl | sh -
$ export PATH=$PATH:$HOME/.istioctl/bin

$ istioctl install --set profile=demo -y

Detected that your cluster does not support third party JWT authentication. Falling
back to less secure first party JWT. See https://istio.io/v1.8/docs/ops/best-
practices/security/#configure-third-party-service-account-tokens for details.
✔ Istio core installed
✔ Istiod installed
✔ Ingress gateways installed
✔ Egress gateways installed
✔ Installation complete
```

Thereafter, install the istio controller and check the installation:

```
$ kubectl apply --filename https://github.com/knative/net-
istio/releases/download/v0.20.0/release.yaml

$ kubectl get pods --namespace istio-system

NAME                                    READY   STATUS    RESTARTS   AGE
istio-egressgateway-8dff9c778-hknw6     1/1     Running   0          176m
istio-ingressgateway-6cfd75fc57-gnwnh   1/1     Running   0          176m
istiod-7f6d7c759-z6hcb                  1/1     Running   0          177m
```

For the purposes of testing, I've ignored the topic of DNS, which is perfectly fine for a private cluster. We'll come back to this shortly, but beforehand, I recommend installing and using the Knative CLI, kn. This in turn requires that golang be installed first, and kn is best built from source:

```
$ git clone https://github.com/knative/client.git
$ cd client/
$ hack/build.sh -f
$ sudo cp kn /usr/bin
$ kn version
Version:      v20210121-local-90c70efe
Build Date:   2021-01-21 20:26:22
Git Revision: 90c70efe
Supported APIs:
* Serving
```

```
  - serving.knative.dev/v1 (knative-serving v0.20.0)
* Eventing
  - sources.knative.dev/v1alpha2 (knative-eventing v0.20.0)
  - eventing.knative.dev/v1beta1 (knative-eventing v0.20.0)
```

And finally after limited pain and effort, we are ready to serve up our first application, with all the benefits of serverless including autoscaling. Here's helloworld:

```
$ kn service create helloworld-go --image gcr.io/knative-samples/helloworld-go --env
TARGET="Go Sample v1"

$ kn service describe helloworld-go
Name:       helloworld-go
Namespace:  default
Age:        49m
URL:        http://helloworld-go.default.example.com

Revisions:
        100%  @latest (helloworld-go-00001) [1] (49m)
        Image:  gcr.io/knative-samples/helloworld-go (at 5ea96b)

Conditions:
        OK TYPE                   AGE REASON
        ++ Ready                  49m
        ++ ConfigurationsReady    49m
        ++ RoutesReady            49m
```

One potential gotcha I found was the need to edit the knative-serving configuration, in order to ignore tag resolution on repos. Launch the editor thusly:

```
$ kubectl -n knative-serving edit configmap config-deployment
```

And then add repos like gcr.io to `registriesSkippingTagResolving`, making sure the updated list is dropped in below the example block where it will be read and used.

Implicit to any Knative app is the need to expose a particular port in your docker container / application / service. Coming back to the issue of DNS or in this case the explicit absence of DNS, knowing our app is using port 80, we find the mapping between external facing and internal ports ala:

```
$ kubectl --namespace istio-system get service istio-ingressgateway

NAME   TYPE    CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
```

```
istio-ingressgateway   LoadBalancer   10.101.90.101   <pending>
15021:30295/TCP,80:30126/TCP,443:30678/TCP,15012:31942/TCP,15443:30866/TCP   21h
```

Armed with this mapping, you can use the hostname header with (for example) curl in order to reach your app eg., here from master:

```
$ curl -H "Host: helloworld-go.default.example.com" http://localhost:30126
Hello Go Sample v1!
```

Summary

This article has briefly detailed installing k8s and Knative, deploying and interacting with a basic service. The next step would likely involve pulling and using a real application, for which you will want to set up and use secrets for accessing a registry (eg., GCR) : https://knative.dev/docs/serving/deploying/private-registry/

On the topic of monitoring, there was at one time two options for doing so, workflows that were very similar to ELK (Elasticsearch + Kibana). However, at the time of writing, links to monitoring solutions no longer exist on the master branch of knative, though you may have some luck with the various forks. Knative is after all fairly new compared to related and more mature platforms like Apache Mesos, so we can expect some volatility yet.

The good news is that major providers like Google and IBM have well documented and highly stable versions of Knative requiring minimal effort to use, complete with monitoring tools and very generous quotas. Tedious tasks like DNS and certificates for https are all managed for you and deployment is a breeze. For example, deploying a service from GCR on Google Cloud Run can be as simple as a single line, or you can use the web console.

For the day job, I developed a solution for offloading many thousands of small physics simulations. This solution is currently hosted on Google Cloud Run, where I take advantage of the ability to launch up to 1k services, each which can run 1k containers simultaneously. The following are some results from a custom dashboard showing statistics for ensembles of simulations, the first when using 64 services to offload up to 1k simulations:

First task start: 2021-01-22T17:32:20.659Z	**Last task start:** 2021-01-22T17:59:23.304Z	**Total tasks:** 903
Min task time (s): 91.193	**Avg task time (s):** 119.0004363233666	**Max task time (s):** 167.525
Compute time (hrs): 29.849276111111124	**Elapsed time (hrs):** 0.4837903989787129	**Compute / elapsed time:** 61.698777350942244
Total outputs: 903	**log(Max obj func):** 5.38093e+0	**log(Min obj func):** 4.55440e+0
Last log msg:	**Task time:** 2021-01-22T17:59:23.304Z	**Task ID:** 92b1c4ef-31cc-4226-969d-a5c47ed6529a
DB Storage (Bytes): 510361600	**DB Objects:** 4499	**DB Health:** 1

128 services :

First task start: 2021-01-22T18:15:08.725Z	**Last task start:** 2021-01-22T18:29:26.121Z	**Total tasks:** 950
Min task time (s): 94.857	**Avg task time (s):** 123.61736526315792	**Max task time (s):** 202.483
Compute time (hrs): 32.62124916666667	**Elapsed time (hrs):** 0.27250371257309935	**Compute / elapsed time:** 119.70937518113993
Total outputs: 950	**log(Max obj func):** 5.38093e+0	**log(Min obj func):** 4.55440e+0
Last log msg:	**Task time:** 2021-01-22T18:29:26.121Z	**Task ID:** 4629d44b-4852-411c-b034-34d40a1102c1
DB Storage (Bytes): 510361600	**DB Objects:** 4499	**DB Health:** 1

And 256 services:

First task start: 2021-01-22T18:56:52.964Z	**Last task start:** 2021-01-22T19:03:41.098Z	**Total tasks:** 977
Min task time (s): 96.757	**Avg task time (s):** 124.83761719549643	**Max task time (s):** 157.6
Compute time (hrs): 33.87954222222223	**Elapsed time (hrs):** 0.14804767144319347	**Compute / elapsed time:** 228.84211478612787
Total outputs: 977	**log(Max obj func):** 5.38093e+0	**log(Min obj func):** 4.55440e+0
Last log msg:	**Task time:** 2021-01-22T19:03:41.098Z	**Task ID:** ad975b9a-1c17-4e71-928c-1e9eb23060ea
DB Storage (Bytes): 510361600	**DB Objects:** 4499	**DB Health:** 1

As you can see, there is some bounce with regards to task computational times, a function of instantaneous server load. However, considering that the cost of ~ 0.1 USD per CPU hour is consistent regardless of how many services are running, and that running 256 services provides at least 200x over a single CPU, the benefits of serverless here are obvious. For more information on Google Cloud Run quotas, please see : https://cloud.google.com/run/quotas

Happy parallel computing! Bill b