Diktyo Guide: How to install the Network-Aware Plugins in K8s

Links and Repositories:

- IEEE TNSM Paper: https://ieeexplore.ieee.org/document/10111024
- KEP approved: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/260-network-aware-scheduling
- PR Accepted: https://github.com/kubernetes-sigs/scheduler-plugins/pull/432
- Diktyo-io: https://github.com/diktyo-io
- Netperf Component: https://github.com/jpedro1992/pushing-netperf-metrics-to-prometheus/tree/configmap

Testbed Requirements:

Kubernetes cluster.
For instance, install Kubernetes via kubeadm v1.24.4 (v1.23.10 also works fine).

Installation Guide:

- 1) Git clone the repository of the Tutorial:

  git clone https://github.com/diktyo-io/tutorial-ieee-netsoft2023.git

  Move to demo-netsoft-2023/ (*cd tutorial-ieee-netsoft2023/demo-netsoft-2023/*)

- 2) Label nodes based on your cluster topology.

  An example is available at *crd-manifests/network-topology-crd/labels.txt* (1 region – 10 zones).

- 3) (Optional) Adding delays to the cluster nodes.

  To simulate a cluster with different delays, please consider adding these via Traffic Control (TC).

  Please check the file tc.txt to see how to add delays to your cluster nodes. Or consider deploying a privileged DaemonSet for making the TC changes.

- 4) Install both CRDs (**Appgroup** and **NetworkTopology**).

  Go to crd-manifests/

  Install AppGroup CRD:

  kubectl apply -f appgroup-crd/appgroup-crd.yaml

  Install Network Topology CRD:

  kubectl apply -f network-topology-crd/network-topology-crd.yaml

  Deploy CRD examples to see that everything works (both used in the demo):

  kubectl apply -f appgroup-crd/online-boutique.yaml
  kubectl apply -f network-topology-crd/nt-cluster.yaml

  Check that both are available in the system:

  kubectl get networktopologies
  NAME AGE
  net-topology-test 1h

kubectl get appgroups
NAME AGE
online-boutique 1h

- 5) Deploy the controllers for both CRDs. Go to network-aware-controllers/

  Create the namespace:

  kubectl apply -f ns.yaml

  Deploy the appGroup controller:
  kubectl apply -f appgroup-controller/

  Deploy the networkTopology controller:

  kubectl apply -f networktopology-controller/

  Please check the pods to see that everything was deployed correctly:

  kubectl get pods -n network-aware-controllers
  NAME READY STATUS RESTARTS AGE
  appgroup-controller-cd49d4546-mwtlr 1/1 Running 0 1h
  networktopology-controller-5fb64c7769-b5j8n 1/1 Running 0 1h

- 6) Deploy the Scheduler and Controller from sig-scheduling (with Network-aware plugins enabled)

  Go to network-aware-scheduler/ and create the namespace:

  kubectl apply -f ns.yaml

  Place the scheduler configuration in the following folder */etc/kubernetes/* or adapt the deploy-scheduler.yaml file

  Install the additional CRDs from sig-scheduling:

  kubectl apply -f extra-crds/
  customresourcedefinition.apiextensions.k8s.io/elasticquotas.scheduling.x-k8s.io created
  customresourcedefinition.apiextensions.k8s.io/podgroups.scheduling.x-k8s.io created
  customresourcedefinition.apiextensions.k8s.io/noderesourcetopologies.topology.node.k8s.io created

  Deploy the controller:

  kubectl apply -f deploy-controller.yaml

  Deploy the scheduler:

  kubectl apply -f deploy-scheduler.yaml

  Please check the pods and logs to see that everything was deployed correctly:

  kubectl get pods -n scheduler-plugins
  NAME READY STATUS RESTARTS AGE
  network-aware-scheduler-d694849b4-fjfvf 1/1 Running 0 1h
  scheduler-plugins-controller-ccbc6dcf5-bnpqg 1/1 Running 0 1h

  kubectl logs scheduler-plugins-controller-ccbc6dcf5-bnpqg -n scheduler-plugins
  I0605 13:11:54.231217 1 logr.go:261] setup "msg"="starting manager"
  I0605 13:11:54.231493 1 elasticquota.go:115] "Starting Elastic Quota control loop"

```
I0605 13:11:54.231596       1 elasticquota.go:117] "Waiting for informer caches to sync"
I0605 13:11:54.231968       1 internal.go:362]  "msg"="Starting server" "addr"={"IP":"::","Port":8081,"Zone":""}
"kind"="health probe"
I0605 13:11:54.232687       1 controller.go:185]  "msg"="Starting EventSource" "controller"="podgroup"
"controllerGroup"="scheduling.x-k8s.io" "controllerKind"="PodGroup" "source"="kind source:
*v1alpha1.PodGroup"
I0605 13:11:54.232716       1 internal.go:362]  "msg"="Starting server" "addr"={"IP":"::","Port":8080,"Zone":""}
"kind"="metrics" "path"="/metrics"
I0605 13:11:54.232792       1 controller.go:185]  "msg"="Starting EventSource" "controller"="podgroup"
"controllerGroup"="scheduling.x-k8s.io" "controllerKind"="PodGroup" "source"="kind source: *v1.Pod"
I0605 13:11:54.232861       1 controller.go:193]  "msg"="Starting Controller" "controller"="podgroup"
"controllerGroup"="scheduling.x-k8s.io" "controllerKind"="PodGroup"
I0605 13:11:54.435311       1 controller.go:227]  "msg"="Starting workers" "controller"="podgroup"
"controllerGroup"="scheduling.x-k8s.io" "controllerKind"="PodGroup" "worker count"=1
I0605 13:11:54.531870       1 elasticquota.go:122] "Elastic Quota sync finished"
```

- 7) Run the Netperf Component:

  Go to the following directory *netperf/*

  First, please install the required dependencies shown in *guide.txt*

  Deploy netperf pods in the cluster:

  kubectl apply -f k8s-netperf.yaml

  Run the scrip *runTestConfigmap.sh*

  Check that the costs are added to the netperf-metrics configmap:

  kubectl get configmaps netperf-metrics

  Running the netperf component only once is enough. Please stop the script and delete all netperf pods.

  kubectl delete -f k8s-netperf.yaml

- 8) Demo: Deploy the OnlineBoutique application with KS and Diktyo and check the performance given by both schedulers.

  Go to *deploy-online-boutique/*

  Deploy Online Boutique with KS:

  kubectl apply -f ks/

  Check that all pods are running before deploying the generator:

  kubectl get pods

NAME	*READY*	*STATUS*	*RESTARTS*	*AGE*
adservice-6584967bbc-mrscz	*0/1*	*Running*	*0*	*11s*
cartservice-6cf6b7755b-njqfv	*0/1*	*Running*	*0*	*11s*
checkoutservice-b85cfd6b7-54njx	*1/1*	*Running*	*0*	*10s*
currencyservice-64ccb66b96-jkvfj	*0/1*	*Running*	*0*	*10s*
emailservice-c9bff4df6-4dh6p	*0/1*	*Running*	*0*	*10s*
frontend-558c8d9d95-l6526	*0/1*	*Running*	*0*	*10s*
paymentservice-6cf97d7fdb-xs2tf	*0/1*	*Running*	*0*	*11s*
productcatalogservice-5d86dd6764-xkwjd	*1/1*	*Running*	*0*	*11s*
recommendationservice-6b54f85898-dzwxr	*1/1*	*Running*	*0*	*11s*
redis-cart-7667674fc7-mp4wb	*1/1*	*Running*	*0*	*10s*
shippingservice-7654b8b7c8-x9492	*1/1*	*Running*	*0*	*10s*

Deploy the load generator:

kubectl apply -f loadgenerator.yaml

Check the performance of OnlineBoutique by checking the logs:

kubectl logs loadgenerator-686c967556-49rsw -f (Check the correct name of the pod)

Name	*# reqs*	*# fails*	*Avg*	*Min*	*Max*	*Median*	*req/s*	*failures/s*
GET /	*17*	*0(0.00%)*	*197*	*64*	*487*	*160*	*0.00*	*0.00*
GET /cart	*11*	*0(0.00%)*	*137*	*88*	*331*	*100*	*0.20*	*0.00*
POST /cart	*12*	*0(0.00%)*	*654*	*172*	*1234*	*650*	*0.60*	*0.00*
POST /cart/checkout	*3*	*0(0.00%)*	*2534*	*1750*	*3155*	*2700*	*0.30*	*0.00*
GET /product/0PUK6V6EV0	*4*	*0(0.00%)*	*154*	*79*	*309*	*110*	*0.10*	*0.00*
GET /product/1YMWWN1N4O	*6*	*0(0.00%)*	*89*	*73*	*132*	*78*	*0.00*	*0.00*
GET /product/2ZYFJ3GM2N	*6*	*0(0.00%)*	*115*	*72*	*210*	*91*	*0.10*	*0.00*
GET /product/66VCHSJNUP	*5*	*0(0.00%)*	*247*	*91*	*633*	*160*	*0.20*	*0.00*
GET /product/6E92ZMYYFZ	*4*	*0(0.00%)*	*90*	*76*	*107*	*86*	*0.00*	*0.00*
GET /product/9SIQT8TOJO	*4*	*0(0.00%)*	*342*	*73*	*771*	*83*	*0.00*	*0.00*
GET /product/L9ECAV7KIM	*4*	*0(0.00%)*	*104*	*88*	*143*	*91*	*0.20*	*0.00*
GET /product/LS4PSXUNUM	*5*	*0(0.00%)*	*96*	*70*	*166*	*78*	*0.00*	*0.00*
GET /product/OLJCESPC7Z	*5*	*0(0.00%)*	*272*	*73*	*914*	*110*	*0.00*	*0.00*
POST /setCurrency	*5*	*0(0.00%)*	*364*	*76*	*845*	*210*	*0.30*	*0.00*
Aggregated	*91*	*0(0.00%)*	*321*	*64*	*3155*	*140*	*2.00*	*0.00*

Delete the load generator and the deployment via KS:

kubectl delete -f loadgenerator.yaml
kubectl delete -f ks/

Deploy Online Boutique now with Diktyo:

kubectl apply -f networkAware/ (the online-boutique CRD was already installed previously in step 2)

Check that all pods are running first, and then deploy the generator again:

kubectl apply -f loadgenerator.yaml

What are the major differences in terms of application performance when deployed with KS or Diktyo?