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

Kube-Scheduler/Scheduler Plugins, <https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/>

Paper: Currently under revision, can be accepted at the time of the assignment.

KEP approved: <https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/260-network-aware-scheduling>

Diktyo-io: <https://github.com/diktyo-io>

Benchmark: Locust and Teastore application (see diktyo/teastore in diktyo.zip)

Testbed: kubeadm cluster v1.22.4 at KUL. At least 5 nodes required per student group.

Guide folder: *diktyo.zip*

**Installation Guide:**

* 1) Install both CRDs. Go to crd-manifests/

Install AppGroup CRD:

*kubectl apply -f appgroup/crd.yaml*

Install Network Topology CRD:

*kubectl apply -f networktopology/crd.yaml*

Deploy CRD examples to see that everything works:

*kubectl apply -f appgroup/example.yaml*

*kubectl apply -f networktopology/networktopology.yaml (also used in the assignment)*

* 2) Label nodes based on the assignment topology. An example is available at *crd-manifests\networktopology/labels.txt* based on the networktopology CR.
* 3) Add random delays between 20 and 60 ms on each ens\* network interface of worker nodes (not the master node!) based on Traffic Control. Please check the file tc.txt to see what is the appropriate tc command. You may not use ssh to login to each node but you need to deploy a DaemonSet for this. Also use an imagePullPolicy so that the container image of the Daemonset is pulled exactly one time from the container registry. You can check that the Traffic Control commands have correctly been executed as follows:

*kubectl exec -it <random pod of daemonset>*

Then the “*ip link show | grep ens.\*”* command must show for all *ens* interfaces the following output:

*2: ens3: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc* ***netem*** *state UFAULT group default qlen 1000*

Delete the Daemonset after on all worker workers nodes these output shows

* 4) Deploy controller:

*kubectl apply -f controller/all-in-one.yaml*

Check logs of the deployed container to see that the following output appears. Please ignore any other error messages that appear in the logs.

*kubectl logs scheduler-plugins-controller-566c9c679f-xzvzd -n scheduler-plugins*

*W0221 09:57:33.693841 1 client\_config.go:615] Neither --kubeconfig nor --master was specified. Using the inClusterConfig. This might not work.*

*I0221 09:57:33.695620 1 appgcroup.go:106] "Starting App Group controller"*

*I0221 09:57:33.695656 1 elasticquota.go:115] "Starting Elastic Quota control loop"*

*I0221 09:57:33.696677 1 elasticquota.go:117] "Waiting for informer caches to sync"*

*I0221 09:57:33.696015 1 podgroup.go:96] "Starting Pod Group controller"*

*I0221 09:57:34.096810 1 elasticquota.go:122] "Elastic Quota sync finished"*

*I0221 09:57:34.096888 1 podgroup.go:103] "Pod Group sync finished"*

*I0221 09:57:34.096908 1 appgroup.go:113] "App Group sync finished"*

* 5) Deploy network-aware-scheduler:

Install it via the deploy-sched-plugins.yaml file. The sched-cc.yaml must be in the master node in the corresponding folder: /etc/kubernetes/

Image: j*pedro1992/kube-scheduler:kubecon*

* 6) Run the Netperf Component:

Go to pushing-netperf-metrics-to-prometheus/

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*

Since delays are added via TC, running the netperf component only once is enough. Please stop the script and delete all netperf pods.

*kubectl delete -f k8s-netperf.yaml*

Note: The netperf component is open-sourced here: <https://github.com/jpedro1992/pushing-netperf-metrics-to-prometheus/tree/configmap>

* 7) Run the teastore application:

Read first *teastore/readme.md*

Deploy the helm chart with normal Kubernetes Scheduler:

*helm install ts teastore/teastore-helm*

Deploy it with Diktyo:

First edit the *teastore/teastore-helm-diktyo/teastore-appgroup-crd.yaml* to add dependencies as shown in the *teastore/architecture.pdf* document with appropriate *maxNetworkCost* and *minBandwidth* fields. Please note that in the architecture document the database component and a number of dependencies has been omitted. These smaller dependencies typically require less bandwidth and can deal with a higher network cost (i.e. traffic control delay).

Then apply the crd

*kubectl apply -f teastore/teastore-helm-diktyo/teastore-appgroup-crd.yaml*

Then, inspect the teastore manifest in *teastore/teastore-helm-diktyo/templates/teastore-clusterip.* What needs to be changed so that the diktyo scheduler is used?

Then delete the helm chart with the normal Kubernetes Scheduler if not already done

*helm delete ts*

Then create the helm chart with the diktyo scheduler

*helm install ts teastore/teastore-helm-diktyo*

To check the performance of Teastire, please run the locust generator as explained in the teastore/readme.md file. This runs a performance experiment during 3 minutes.

**What differences/conclusions can you already draw from these? Is the performance of Diktyo as expected? What can you change to improve it?**