# Lab-01: Explore the setup

* see cluster setting

kubectl get nodes

* see more details of the cluster (including IP address, k8s version, container-Runtime with version)

kubectl get nodes -o wide

* see at what IP address kube-api-server is running

kubectl cluster-info

* ~~if you want more information about everything on the cluster (it will produce lots of info)~~

~~kubectl cluster-info dump |less~~

* see existing namespaces

kubectl get ns

kubectl get namespaces

* How do we know that “ns” is a short name for “namespaces”?

kubectl api-resources

it will show all resources and their short names (if any), for example:

pods => po, nodes => no, persistenceVolumeClaim => pvc, services => svc

* see existing pods

kubectl get pods

why is it not showing any pods?

All commands get the resources from the given namespace only. Namespace ‘default’ is used if no namespace is provided, and no context is set. There are no pods in the default namespace.

* see existing pods in kube-system namespace

kubectl get pods -n kube-system

to see more details (including on which node each pod is running)

kubectl get pods -n kube-system -o wide

deleting the pod will be respawned by k8s (on the same or different node)

Please note the name of one of the “coredns” pods, its IP address, and on which node that pod is running

**For example:**

coredns-6d4b75cb6d-g47z9, running on worker02, IP:10.244.2.76

* delete this pod (noted above)

kubectl delete pod -n kube-system <name of the pod>

* it will be respawned by k8s. see if a new pod is scheduled on the same or on a different node. This will have a new IP address.

kubectl get pods -n kube-system -o wide

‘coredns’ new pod will have a slightly different name. You can see the newly created pod by looking at the age of ‘coredns’ pods.

* See all pods from all namespaces

**Note:** Compare all these commands, all these are equivalent

kubectl get pods --all-namespaces

kubectl get pods -A

kubectl get po -A

* See all resources (e.g., pods, services, deployment, daemonsets, etc.) running by the system itself.

**Note:** Under the NAME column, resource type (pod, service, daemonset, etc.) is listed before the resource name.

kubectl get all -A

kubectl get all -A -o wide # it will show more details

**END OF LAB-01**

# Lab02 (run the pods)

* Create a namespace for lab02 (use any one of these two commands)

kubectl create namespace ns-lab02 #<<< long version

kubectl create ns ns-lab02 #<<< short version

* Create a pod in this namespace. Nginx is a public image available in the docker hub (public registry).

kubectl run ngx-testpod --image=nginx -n ns-lab02

* View this pod

kubectl get pods

* Nothing is listed by the previous command, as we did not specify the correct namespace. Now try the same command with correct namespace.

kubectl get pods -n ns-lab02

* View allocated IP and node

kubectl get pods -n ns-lab02 -o wide

each pod always gets one IP address allocated by k8s. In this case, we cannot use this IP for anything as we did not expose any port on our pod. Hence pod will have IP address allocated but we cannot interact with the pod for any practical purpose as there is no open port. We know nginx listens on ports 80 and 443 but k8s also needs to expose these ports before we can access nginx ports.

Note: Other Pods running in the same namespace as nginx pod, can access the nginx pod’s port 80 and 443 by using nginx-pod’s IP address. This does not have much practical use. Once the pod restarts it will get new IP and we have no easy method to know the new IP and access the pod’s port at that new IP.

* Start a buxybox pod

**Note:** busybox is a very popular docker image used as an easily available debug utility. There are many such public images that exist. Another popular image is ‘alpine’. It has various Linux utilities (ping, wget, etc) included in it.

kubectl run busybox-pod --image=busybox

see if this is running? If not, why?

Busybox (or any container) is like a unix process that exits as soon as the command is completed. We did not provide any command, so it did exit immediately. We need to attach input (and output ‘tty’) so that shell always wait for user inputs. K8s will keep restarting it and it will keep exiting (completed).

Also, we did not provide any namespace, which we should do for all activities related to lab02.

* Delete that completed pod we tried creating incorrectly.

kubectl delete pods busybox-pod

* Now create the pod correctly

kubectl run -i --tty busybox-pod --image=busybox -n ns-lab02 # we are attaching input and output to shell

* Use *ls -l /bin/* to see what unix utilities are available in busybox image.
* Try a few available Unix commands such as *ls,* *cat*, *ping*, *pwd,* etc.
* type *exit* or enter *ctrl-D* to come out of busybox shell
* Reattach that pod to interact with it

kubectl attach busybox-pod -i -t -n ns-lab02

* type *exit* or enter *ctrl-D* to come out of it
* delete everything we did for lab02 by deleting the namespace ns-lab02. It will also delete whatever we have created/deployed in that namespace

kubectl get pods -n ns-lab02 # see what pods are running

kubectl delete ns ns-lab02 # delete this namespace

# Lab 03 (Logs and description)

* Create namespace ns-lab03

kubectl create ns ns-lab03

* In our setup, k is the alias of kubectl, so going forward we will use “k” instead of “kubectl”. You can use kubectl instead of ‘k’ if you prefer so.

k get namespaces # notice the newly created ns in this list

* create nginx pod

k run my-nginx-pod --image=nginx -n ns-lab03

* see logs and analyze these

k logs my-nginx-pod -n ns-lab03

* describe the pod and see various sections. The events section provides very useful information for debugging.

k describe po my-nginx-pod -n ns-lab03

* we can login to any container using this syntax. Containers in pods are just unix processes. Since this simple pod has just one container, we do not need to provide a container name. In multi-container pods, you must also provide the container name to be login.

k exec -it my-nginx-pod -n ns-lab03 -- bash # bash may not be available in all images, try ‘sh’ in that case

* try basic unix commands here. You can access “index.html” file at “/usr/share/nginx/html”

*cd /usr/share/nginx/html*

*cat index.html*

* type *exit* or *Ctrl-D* to logout
* We need to start a busybox pod to access nginx.

k run -i --tty busybox-pod -n ns-lab03 --image=busybox -- sh

* exit from busybox by typing ‘*exit’* or *Ctrl-D*.
* We are tired of typing namespace=ns-lab03, we can set the context so we do not need to type it anymore.

k config set-context --current --namespace=ns-lab03

* Now all commands will use ns-lab03 namespace by default (unless you explicitly provide another namespace).

k get pods -o wide

you will see pods from ns-lab03 namespace. Please note down the IP address of my-nginx-pod

* Attach to shell in busybox pod.

k attach busybox-pod -it

* use wget in the shell to access the nginx default webpage (index.html, which we saw earlier). We are here accessing nginx pod using its k8s allocated IP address.

*wget -qO- <IP address of my-nginx-pod>*

Note: if my-nginx-pod gets restarted, it will get new IP and you need to adjust wget command above (i.e., the command working now may not work after some time if the pod is restarted or rescheduled on a different worker node).

* Exit the busybox by typing *exit* or *Ctrl-D.*
* delete everything we did for lab03 by deleting the namespace ns-lab03. It will also delete whatever we have created in that namespace

k delete ns ns-lab03

# Lab 04 (Deployments)

* Create lab04 namespace

k create ns ns-lab04

* set the context so we do not need to specify the namespace

k config set-context --current --namespace=ns-lab04

* deployments ensure that you can progressively deploy, roll back, and scale your applications without downtime.

k create deployment ngx-dep --image=nginx --replicas=6

k get pods # will show 6 nginx pods running

* delete a pod or two and k8s will keep the replicas to 6.

k delete po <pod-name>

Note: If a worker node gets rebooted, pods from that worker node will be started on some other worker node to maintain the replica count to 6

* Scale the replicas to 10

k scale deployment ngx-dep --replicas=10

* Notice that now 10 pods are running.

k get pods

* see these pods are distributed across on all 3 worker nodes

k get po -o wide --sort-by=.spec.nodeName

* Scale the replicas to 6 and observe that only 6 instances are running now.

k scale deployment ngx-dep --replicas=6

* Are these 6 pods distributed evenly across all worker nodes?

k get po -o wide --sort-by=.spec.nodeName

* Scale the replicas to 0 and observe that no instances are running now.

k scale deployment ngx-dep --replicas=0

why do we need –replicas=0? if we need to stop the application but keep all the configuration intact or need to bounce all pods at once. We can set it to 0 and then back to the desired number. There may be other use-cases too.

* Delete the lab04 and reset the context from ns-lab04.

k config set-context --current --namespace=default

k delete ns ns-lab04 --force # use force to delete quickly

# Lab 05 (Working with yaml)

* Create lab05 namespace

k create ns ns-lab05

* set the context so we do not need to specify the namespace with every command

k config set-context --current --namespace=ns-lab05

* whatever we did so far is good for learning. In the production environment, everything needs to be done using configuration files and those files likely will be source controlled.
* Review the file ~/lab05/lab05-manually-created-for-pod.yaml and Deploy a simple pod using yaml.

k apply -f ~/lab05/lab05-manually-created-for-pod.yaml

* Delete this pod using yaml (you can also delete the pod using the kubectl command as we did in the earlier lab)

k delete -f ~/lab05/lab05-manually-created-for-pod.yaml

* Run this pod using the imperative command

k run my-ngx-pod --image=nginx --port=80

* Create yaml from this running pod

k get po my-ngx-pod -o yaml > ~/lab05/lab05-created-from-pod-output.yaml

**Review this yaml file** ~/lab05/lab05-created-from-pod-output.yaml to see how much configuration and status are involved in a simple pod. **Compare it with** ~/lab05/lab05-manually-created-for-pod.yaml which we used earlier. Both versions basically will serve the same purpose.

* Since we have yaml, we can delete the pod and recreate the same using this generated YAML

k delete po my-ngx-pod

and now recreate the same pod using the yaml generated earlier by this same pod.

k apply -f ~/lab05/lab05-created-from-pod-output.yaml

**Note:** yaml generated by kubectl has lots of metadata as well as default values which we do not require to provide while creating our own yaml. K8s will use default values for whatever is not included in yaml.

* Also, View JSON from running pod

k get po my-ngx-pod -o json |jq

**Note: we can always use json instead of yaml**. Yaml is more common and there is no technical reason for not using json instead. We can have a few files in JSON and the rest in yaml too.

**For example:**

Copy configuration in json instead of yaml format

k get po my-ngx-pod -o json > ~/lab05/lab05-created-from-pod-output.json

delete the pod

k delete po my-ngx-pod

Recreate the pod using json

k apply -f ~/lab05/lab05-created-from-pod-output.json

**Now we move to next exercise**

* Review the file ~/lab05/lab05-02-deploy.yaml and create the deployment

k apply -f ~/lab05/lab05-02-deploy.yaml

k get po

See that there are 6 replicas of nginx. Now manually change the replicas from 6 to 10 in file ~/lab05/lab05-02-deploy.yaml by using a text editor.

* Apply the newly modified YAML.

k apply -f ~/lab05/lab05-02-deploy.yaml

k get po

Note that there are 10 replicas of nginx pods now.

* Edit the running yaml, change replicas from 10 to 8. Do not change anything else. As soon as you save this configuration, it will be applied.

k edit deployments.apps nginx-deployment

k get po

* Apply the YAML (which has replicas=10). It will increase replicas to 10.

k apply -f ~/lab05/lab05-02-deploy.yaml

k get po -o wide

All these ‘get’ command outputs are also including our old pod “my-ngx-pod”. If we need to list only pods which belong to this deployment. One way to filter desired pods is using the selector.

k get pod --selector=my-pod-label=nginx-abcd

Now review the file ~/lab05/lab05-02-deploy.yaml and determine what we did here.

Labels are heavily used in Kubernetes to select the desired resources. You apply labels to any resources and use the selector to filter based on those labels.

* Delete the lab05 and reset the context from ns-lab05.

k config set-context --current --namespace=default

k delete ns ns-lab05 --force # use force to delete quickly

# Lab 06 (Labels and Selectors)

* Create lab06 namespace

k create ns ns-lab06

* set the context so we do not need to specify the namespace with every command

k config set-context --current --namespace=ns-lab06

Labels can be added to any resource. There is no limit to how many labels you apply to any resource.

* Check if any label exists in the namespace resource.

k describe ns ns-lab06 # look for labels

* Create a pod and apply label.

k run my-nginx-pod1 --image=nginx -l key1=value1

* View the label using

k describe po my-nginx-pod1

* Apply multiple labels

k run my-nginx-pod2 --image=nginx -l key1=value1,key2=value2

k describe po my-nginx-pod2

* each key is unique, if you use multiple copies of the same key, the last one will be effective.

k run my-nginx-pod3 --image=nginx \

-l key1=value1,key2=value2,key2=value3

k describe po my-nginx-pod3

Note: Label key2=value3 will be used.

* each key is unique, value is optional (note that there is no value for key2)

k run my-nginx-pod4 --image=nginx -l key1=value1,key2=,xyz=abcd

k describe po my-nginx-pod4

* list pods based upon label(s)

k get po -l key1=value1 # all 4 pods will be listed

k get po -l key2=value3 # only one pod will be listed

k get po -l key2 # key2 exists with any value (or no value)

k get po -l key2 -l key1 # key1 or key2 exists

k get po --selector=key1,key2 # same as above but diff syntax

**Note:** any number of labels can be applied to any Kubernetes resource.

* Delete the lab06 and reset the context from ns-lab06.

k config set-context --current --namespace=default

k delete ns ns-lab06 --force # use force to delete quickly

# Lab 07 (Services)

A Kubernetes service is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function). Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, database, etc.) to be assigned a name and unique IP address.

**Service types**: nodePort, clusterIP, loadBalancer, externalName

Not to be confused with deployment.

A deployment is responsible for keeping a set of pods running.

A service is responsible for enabling network access to a set of pods.

* Create lab07 namespace

k create ns ns-lab07

* set the context so we do not need to specify the namespace with every command

k config set-context --current --namespace=ns-lab07

* **Analyze/Review** the file ~/lab07/lab07-apache-deploy.yaml and then apply this configuration to create the webserver deployment

k apply -f ~/lab07/lab07-apache-deploy.yaml

k get deployments my-apache-deployment

k get po

* **Analyze/Review** the file ~/lab07/lab07-apache-service.yaml and then apply this configuration to create the service

k apply -f ~/lab07/lab07-apache-service.yaml

k get service apache-http-service

* Issue following commands and see IP addresses or service endpoints are matching with those of pods in ‘my-apache-deployment.

k get po -l app=webserver -o wide

k get endpoints apache-http-service

Please Compare IP addresses.

* Now delete the deployment and see how many endpoints are there? Is IP address of the service is still there and intact?

k delete -f ~/lab07/lab07-apache-deploy.yaml

k get svc apache-http-service # svc IP will never change

k get ep apache-http-service # no endpoints now

* Redeploy the apache webservers and endpoints will reappear. This time all pods will have new IP and service will add correct IP as service-endpoints.

k apply -f ~/lab07/lab07-apache-deploy.yaml

k get po -l app=webserver -o wide

k get endpoints apache-http-service

* Scale the deployment to 10 replicase and observe the service end points..

k scale deployment my-apache-deployment --replicas=10

k get po -l app=webserver -o wide

k get ep apache-http-service # not much space to display all IP

k describe ep apache-http-service # now see all 10 ep IPs

* Run a busybox pod so we can access the service using the IP address. We will access the service IP and the service will send the request to one of the backend pods (typically round-robin load-balancing).

k get svc apache-http-service # Please note svc Cluster-IP & port

k run -it my-busybox-pod --image=busybox

wget -qO- <service IP>:8080

If we delete and recreate the service, it will be allocated a new IP. This is not easy to keep track of in a production environment. To make it easy, Kubernetes automatically creates FQDN for all resources. So if IP is changed due to restart, FQDN will never change.

FQDN for service is <svc-name>.<name space>.svc.cluster.local

Now try the same command using FQDN

wget -qO- apache-http-service.ns-lab07.svc.cluster.local:8080

* Now you can delete and recreate the service, which will result in a new cluster IP. But this FQDN will be intact.

Delete/recreate the deployment (it will create new pod Ips)

k delete -f ~/lab07/lab07-apache-deploy.yaml

k apply -f ~/lab07/lab07-apache-deploy.yaml

Delete/recreate the service (it will allocate a different IP for the service)

k delete -f ~/lab07/lab07-apache-service.yaml

k apply -f ~/lab07/lab07-apache-service.yaml

Observe new pod Ips and new service IP.

k get po -l app=webserver -o wide

k get svc apache-http-service

k get ep apache-http-service # all pods are added as ep

access busybox pod and run the command we used earlier

k attach -it my-busybox-pod

wget -qO- apache-http-service.ns-lab07.svc.cluster.local:8080

**Note:** we got new pod IPs, and new service IPs but a user is still able to access it without worrying about the ephemeral nature of all k8s resources.

* Delete the lab07 and reset the context from ns-lab07.

k config set-context --current --namespace=default

k delete ns ns-lab07 --force