Tips

- Use `--dry-run=client` to create a pod template as the starting point. Example: `k run webapp --image=my-webapp --dry-run=client -o yaml > webapp.yaml`

  This will create `webapp.yaml` with the following contents:

  ```
  apiVersion: v1 kind: Pod metadata: creationTimestamp: null labels: run: w
  ebapp name: webapp spec: containers: - image: my-webapp name: webapp reso
  urces: {} dnsPolicy: ClusterFirst restartPolicy: Always status: {}
  ```

- Use the following tool to quickly switch between contexts and namespaces: **kubectx**
  - Run `brew install kubectx` to install both `kubectx` and `kubens`
  - `kubens` - switch to a namespace in a dropdown (requires `fzf` to be installed)
- We can install plugins from Installing Addons | Kubernetes to add functionalities that are not natively provided by K8s. These include networking solutions, service discovery etc.

Internal Workings

Kubernetes network stack fundamentals: How pods on di...

Learn how pods communicate with each other when they are on different Kubernetes nodes.

https://www.redhat.com/sysadmin/kubernetes-pods-commun...

A Guide to the Kubernetes Networking Model

Kubernetes was built to run distributed systems over a cluster of machines. The very nature of distributed systems makes networking a central and necessary component of Kubernetes deployment, and understanding

https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/

Demystifying kube-proxy

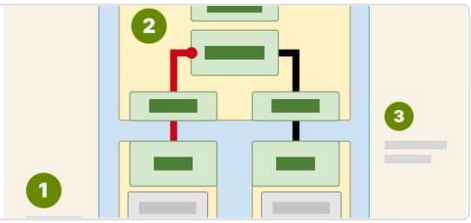
This blog post explains how the kube-proxy component of Kubernetes works internally. It goes through various laters of networking abstractions in a top-down approach, ultimately leading to kube-proxy.

https://mayankshah.dev/blog/demystifying-kube-proxy/#extra-reading

Tracing the path of network traffic in Kubernetes

Learn how packets flow inside and outside a Kubernetes cluster. Starting from the initial web request and down to the container

https://learnk8s.io/kubernetes-network-packets



How declarative commands work?

The declarative `kubectl apply` command checks the **local config file** (stored on the local system), the **live config** (stored in K8s memory) and the **last applied config** (stored in the live config as an annotation in JSON format), to decide what changes are to be made to take the system to the desired state.

```
.yaml
ersion: v1
: Pod

data:
e: myapp-pod
els:
app: myapp
type: front-end-service
:
ntainers:
name: nginx-container
image: nginx:1.18
```

```
ectl apply -f nginx.yaml
```

```
Live object configuration
apiVersion: v1
kind: Pod

metadata:
 name: myapp-pod
 annotations:
  kubectl.kubernetes.io/last-applied-configurat
 {"apiVersion": "v1","kind": "Pod","metadata": {"annotatio
 {},"labels": {"run": "myapp-pod","type": "front-end-servi
     },"name": "myapp-pod"},"spec": {"containers": [{"imag
 "nginx:1.18","name": "nginx-container"}]}}
 labels:
     app: myapp
     type: front-end-service
spec:
  containers:
  - name: nginx-container
    image: nginx:1.18
status:
```

Last applied config is required to find out if something has been removed in the local config file.


Local file
Last applied Configuration
Kubernetes
Live object configuration
apiVersion: v1
kind: Pod
metadata:
name: myapp-pod
labels:
app: myapp
spec:
containers:
- name: nginx-container
image: nginx:1.19
status:
conditions:
"apiVersion": "v1",
"kind": "Pod",
"metadata": {
"annotations": {},
"labels": {
"run": "myapp-pod",
"type": "front-end-service"
},
"name": "myapp-pod",
},
"spec": {
"containers": [
{
"image": "nginx:1.19",
"name": "nginx-container"
}
]
}
}

CKA

- Kubectl Reference - Kubectl Reference Docs (kubernetes.io)

Wrong questions

- Pod to print something at regular intervals

```
apiVersion: v1 kind: Pod metadata: creationTimestamp: null labels: run: l
ooper-cka16-arch name: looper-cka16-arch spec: containers: - image: busyb
ox name: looper-cka16-arch command: - "sh" - "-c" - "while true; do echo
hello; sleep 10; done"
```

- PVC cannot be edited, it must be deleted and recreated
- Create ClusterRoleBinding for a ServiceAccount

```
kubectl create clusterrolebinding <clusterrolebinding-name> --clusterrole
=<clusterrole-name> --serviceaccount=<namespace>:<serviceaccount-name>
```

- SC, PV and PVC with full options

```
apiVersion: storage.k8s.io/v1 kind: StorageClass metadata: name: orange-s
tc-cka07-str provisioner: kubernetes.io/no-provisioner volumeBindingMode:
WaitForFirstConsumer --- apiVersion: v1 kind: PersistentVolume metadata:
name: orange-pv-cka07-str spec: capacity: storage: 150Mi volumeMode: File
system accessModes: - ReadWriteOnce persistentVolumeReclaimPolicy: Retain
storageClassName: orange-stc-cka07-str local: path: /opt/orange-data-cka0
7-str nodeAffinity: required: nodeSelectorTerms: - matchExpressions: - ke
y: kubernetes.io/hostname operator: In values: - cluster1-controlplane --
- apiVersion: v1 kind: PersistentVolumeClaim metadata: name: orange-pvc-c
ka07-str spec: accessModes: - ReadWriteOnce volumeMode: Filesystem resour
ces: requests: storage: 128Mi storageClassName: orange-stc-cka07-str volu
meName: orange-pv-cka07-str
```

- Using `jsonpath` to get the value of a key in YAML

```
kubectl --context cluster1 get pod <podname> -o jsonpath='{.metadata.labels.stack}'
```

- Take ETCD backup using `etcdctl`

```
ETCDCTL_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /opt/cluster1_backup.db
```

- DNS name of a pod - `<pod-ip-separated-by-hyphens>.<namespace>.pod`
- Check if a user has access to perform an operation (`User` is not namespace bound)

```
k auth can-i get deployments --as <user-name>
```

- Check if a service account has access to perform an operation

```
k auth can-i get deployments --as system:serviceaccount:<namespace>:<serviceaccount-name>
```

- If a PVC is not getting bounded to a PV, check if the PV has `claimRef` referring to a specific version of the PVC.
- Display the name and IP address of all the pods in a namespace

```
kubectl get pods -n <namespace> -o=custom-columns='POD_NAME:metadata.name,IP_ADDR:status.podIP' --sort-by=status.podIP
```

- Ingress resource with SSL redirect turned off

```
apiVersion: networking.k8s.io/v1 kind: Ingress metadata: name: nginx-ingr
ess-cka04-svcn annotations: nginx.ingress.kubernetes.io/ssl-redirect: "fa
lse" spec: rules: - http: paths: - path: / pathType: Prefix backend: serv
ice: name: nginx-service-cka04-svcn port: number: 80
```

- When mounting a config file as a configMap type volume, mount the volume at the directory containing the config file (not the full path of the config file).
- Config files for `kubelet` are present under `/var/lib/kubelet` on every node.
- Endpoint object to send traffic to an external web server

```
apiVersion: v1 kind: Endpoints metadata: # the name here should match the
name of the Service name: external-webserver-cka03-svcn subsets: - addres
ses: - ip: <node-ip> ports: - port: 9999
```

- `apiGroups: ""` (core) for namespace as resource in a cluster role.
- If `kube-apiserver` is not reachable or not running, grep into the kubelet logs using `journalctl -u kubelet | grep` command and look closely for error logs.
- `spec.volumeName: <pv-name>` to make a PVC request storage from a specific PV
- Check the details like image name of a deployment revision - `k rollout history deployment <deployment-name> --revision <revision-number>`
- If pods are not able to resolve a hostname like `example.com`, it's likely because CoreDNS containers in `kube-system` namespace is down. In this case if you see no CoreDNS containers, it's possible that the `coredns` deployment in `kube-system` namespace is scaled down to `0`. The solution in this case would be to scale the `coredns` deployment to `2`.

- Allowing volume readonly permission to the sidecar container

```
apiVersion: apps/v1 kind: Deployment metadata: name: olive-app-cka10-str
spec: replicas: 1 template: metadata: labels: app: olive-app-cka10-str sp
ec: containers: - name: python ... volumeMounts: - name: python-data moun
tPath: /usr/share/ - name: busybox ... volumeMounts: - name: python-data
mountPath: "/usr/src" readOnly: true
```

- Logs for stopped containers like `kube-apiserver` can be found at `/var/log/pods`. This is specially useful when `kube-apiserver` is down and we need it's container logs.
- If several pods in the `kube-system` namespace are crashing repeatedly after some time, a possible cause can be `kube-apiserver` pod crashing due to wrong health check configuration.
- Having `spec.ingressClassName` in `Ingress` is important. If it's not specified in the question, find the ingress class and specify it.
- To match all the pods, set `spec.podSelector: {}`
- `NetworkPolicy` to only allow pods in `space1` to reach pods in `space2`

  By default, all the TCP ports are open in a network policy. We need to explicitly allow port 53 on UDP so that pods in `space1` can resolve DNS queries.

```
apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: np n
amespace: space1 spec: podSelector: {} policyTypes: - Egress egress: - t
o: - namespaceSelector: matchLabels: kubernetes.io/metadata.name: space2
- ports: - protocol: UDP port: 53 --- apiVersion: networking.k8s.io/v1 ki
nd: NetworkPolicy metadata: name: np namespace: space2 spec: podSelector:
{} policyTypes: - Ingress ingress: - from: - namespaceSelector: matchLabe
ls: kubernetes.io/metadata.name: space1
```

- RBAC Rules

  A `ClusterRole` / `Role` defines a set of permissions and where it is available, in the whole cluster or just a single namespace.

  A `ClusterRoleBinding` / `RoleBinding` connects a set of permissions with an account and defines where it is applied, in the whole cluster or just a single namespace.