# Kubernetes WSL Ubuntu 24.04

2024-08-08T16:15:50.819+02:00

These instructions will enable you to run `podman`, `kubectl` and `minikube` on Windows 11 using WSL2 with an `Ubuntu 24.04` distribution. Please notice that if you ever mess up the installation or you just somehow encounter errors or strange behavior you can stomp the entire distribution from `powershell` using the command `$> wsl --unregister ubuntu` (this also works for every other distribution name if necessary).

## Set Up Instructions

1. Download *ubuntu-24.04* as WSL on the microsoft store and create `~/.wslconfig` like so

```
[wsl2]
memory=20GB
processors=6
localhostForwarding=true
dnsProxy=true
autoProxy=true
nestedVirtualization=true
```

2. Open the distribution by typing `wsl` in powershell and setup user name and password

3. Install podman and necessary components using following statements

```
sudo apt-get update
sudo apt-get -y install podman
sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils
sudo systemctl enable libvirtd
sudo systemctl start libvirtd
```

4. Install `gvproxy` (look up the newest version of the gvproxy-repo and replace `v0.7.4`)

```
cd /tmp
sudo wget https://github.com/containers/gvisor-tap-vsock/releases/download/v0.7.4/gvproxy-linux-amd64 \
    -O /usr/libexec/podman/gvproxy
sudo chmod +x /usr/libexec/podman/gvproxy
```

5. Open the file `~/.config/containers/containers.conf` and edit the `[engine]` stanza

```
[engine]
helper_binaries_dir = ["/usr/libexec/podman"]
# … some more automatically generated content will already be in here no worries and do NOT delete that …
```

6. Give permissions on `/dev/kvm` to every user on your system *(this is discouraged on mission critical systems)*

```
sudo chmod 777 -R /dev/kvm
```

7. Make the `/` mount shared *(this is discouraged on mission critical systems)*

```
sudo mount --make-rshared /
```

8. Start up podman with following commands

```
# if you do not restart, you can ommit the first two statements
podman machine stop
podman machine rm -f
podman machine init --cpus 6 --disk-size 256 --memory 8192
podman machine set --rootful
podman machine start
```

10. Install *kubernetes cli* executing following commands

```
cd /tmp
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
# [output] > kubectl: OK
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl && rm kubectl && rm kubectl.sha256
kubectl version
```

*This will complain that the connection to the server was refused don't worry.*

11. Install *minikube* using the same method

```
cd /tmp
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64.sha256
echo "$(cat minikube-linux-amd64.sha256)  minikube-linux-amd64" | sha256sum --check
# [output] > minikube-linux-amd64: OK
sudo install -o root -g root -m 0755 minikube-linux-amd64 /usr/local/bin/minikube
rm minikube-linux-amd64 && rm minikube-linux-amd64.sha256
minikube version
```

*This will complain again don't worry.*

12. Go outside take a deep breath, everything will be okay

13. To be able to actually run `minikube` you have to update the `sudoers` file

```
whoami
# [output] > <your-username>
sudo visudo
# add an entry at any position on any line the order does not matter
<your-username> ALL=(ALL) NOPASSWD: /usr/bin/podman
```

14. Now you can actually start `minikube`

```
minikube start --driver=podman --container-runtime=cri-o --network=host
# [output] > Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

*If you do not get the "Done!" in the output don't worry. It means you will go through "google-duckduckgo-ChatGPT-hell" now. This will be an agonizing experience but it does not make it any better if you worry.*

> *To reset minikube or stop the service run:*

```
minikube stop
minikube delete --all
minikube start --driver=podman --container-runtime=cri-o --network=host
```

## *Create Deployment*

In this section we create a deployment directly on the command line. This method is not recommended for several reasons but it shows how all of the deployment steps take place.

1. Check pods and services

```
kubectl get nodes
# [output]
# NAME       STATUS   ROLES           AGE   VERSION
# minikube   Ready    control-plane   29m   v1.30.0

minikube status
# [output]
# minikube
# type: Control Plane
# host: Running
# kubelet: Running
# apiserver: Running
# kubeconfig: Configured

kubectl get pod
# [output] > No resources found in default namespace.

kubectl get services
# [output]
# NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
# kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   5m10s
```

2. Make sure you have access to an image for a pod *(either by using a Containerfile or by accessing an online repository. In this example we use `quay.io/coreos/alpine-sh`)*

3. Create the deployment

```
# working test with a blocked and sleeping pod that will log out "Log from the pod" and goes to sleep for 3 minutes
kubectl create deployment my-deployment-name --image=quay.io/coreos/alpine-sh -- /bin/sh -c "echo Log from the pod && sleep 180"
```

4. Check pod (output deployment-replicahash-podid)

```
kubectl get pod
# [output]
# NAME                                READY   STATUS    RESTARTS   AGE
# my-deployment-name-58d79b7854-xn7r8   1/1     Running   0          49s
```

5. Check replica set (output deployment-replicahash)

```
kubectl get replicaset
# [output]
# NAME                           DESIRED   CURRENT   READY   AGE
# my-deployment-name-58d79b7854   1         1         1       6m6s
```

6. Check deployment

```
kubectl get deployment
# [output]
# NAME                READY   UP-TO-DATE   AVAILABLE   AGE
# my-deployment-name   1/1     1            1           22m
```

## *Update, Monitor and Delete Deployment*

1. Edit deployment

```
kubectl edit deployment my-deployment-name
# [output] > status-file content as vi-session
# [note] > this is 'vi' press 'i' to insert, the 'esc' to exit insert mode
# and then ':q!' to cancel editing or ':wq!' to write and quit
```

2. Try updating the number of replicaset from 1 to 2 and check the deployment again

```
kubectl get replicaset
# [output]
# NAME                          DESIRED   CURRENT   READY   AGE
# my-deployment-name-58d79b7854   2         2         2       117s
```

3. Print the pods logs *(the logs have to be printed while the pod is in 'running' state)*

```
kubectl logs my-deployment-name-58d79b7854-xn7r8
# [output] > Log from the pod
```

4. Start an interactive terminal on the container

```
kubectl exec -it my-deployment-name-58d79b7854-xn7r8 -- bin/sh
# [output] > You get to use a bash session on the pod and can use linux commands like 'ls -al'
```

5. Delete the deployment

```
kubectl delete deployment my-deployment-name
# [output] > deployment.apps "my-deployment-name" deleted
```

## *Create Deployment From YAML*

In this section a yml file will be created to deploy a `docker.io/library/nginx` image. Make sure that your computer has internet access and the command `podman pull docker.io/library/nginx` actually works as expected without any errors.

1. Set up a configuration file (for example deployment-config.yml) for your deployment with content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mr-black # choose this name freely
  labels:
    key: black-app # this can be literally any key-value
spec:
  replicas: 2 # number of pods containing the same image
  selector:
    matchLabels:
      key: black-app
  template:
    metadata:
      labels:
        key: black-app
    spec:
      containers:
      - name: black-app
        image: docker.io/library/nginx:1.16
        ports:
        - containerPort: 5645
```

2. To create the deployment run

```
kubectl apply -f ./deployment-config.yml
# [output] > deployment.apps/mr-black created
```

## *Create Service For Deployment*

1. To add a service create a file (for example service-config.yml) with content:

```
apiVersion: v1
kind: Service
metadata:
  name: mr-black-service # choose this name freely
spec:
  selector:
    key: black-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5645 # has to match deployment containerPort
```

2. To create the service run

```
kubectl apply -f ./service-config.yml
# [output] > service/mr-black-service created
```

3. To get the `status` that is stored in `etcd` key-value store of kubernetes run

```
kubectl get deployment mr-black -o yaml > status.yml
```

4. To check everything is running

```
kubectl get all
# [output]
# NAME                            READY   STATUS    RESTARTS   AGE
# pod/mr-black-7c6b98f888-f5w9r   1/1     Running   0          2m51s
# pod/mr-black-7c6b98f888-mkv6l   1/1     Running   0          2m51s

# NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
# service/kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP   3h35m
# service/mr-black-service   ClusterIP   10.100.217.246   <none>        80/TCP    62s

# NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
# deployment.apps/mr-black   2/2     2            2           2m51s

# NAME                                  DESIRED   CURRENT   READY   AGE
# replicaset.apps/mr-black-7c6b98f888   2         2         2       2m51s
```

5. To remove everything run

```
kubectl delete service mr-black-service
# [output] > service "mr-black-service" deleted
kubectl delete deployment mr-black
# [output] > deployment.apps "mr-black" deleted
```

## Namespaces

Not only the `name` property of every service, pod, secret and configmap can be set seperately and dynamically, also the `namespace` property can be set. This is not set randomly but usually created first.

You can do this by running:

```
kubectl create namespace black-space
# [output] > namespace/black-space created
```

Or you can use a config file for example like this:

```
apiVersion: v1
kind: Namespace
metadata:
  name: black-space
```

However `Volumes` and `Nodes` can not be created in a namespace. They live *globally* in a cluster and they can not be *isolated*. Nevertheless you can check if there are resources like that with following command:

```
kubectl api-resources --namespaced=false
# [output]
# NAME                              SHORTNAMES   APIVERSION                        NAMESPACED   KIND
# componentstatuses                 cs           v1                                false        ComponentStatus
# namespaces                        ns           v1                                false        Namespace
# nodes                             no           v1                                false        Node
# persistentvolumes                 pv           v1                                false        PersistentVolume
# mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1   false        MutatingWebhookConfiguration
# validatingadmissionpolicies                    admissionregistration.k8s.io/v1   false        ValidatingAdmissionPolicy
# validatingadmissionpolicybindings              admissionregistration.k8s.io/v1   false        ValidatingAdmissionPolicyBinding
# validatingwebhookconfigurations                admissionregistration.k8s.io/v1   false        ValidatingWebhookConfiguration
# customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1           false        CustomResourceDefinition
# apiservices                                    apiregistration.k8s.io/v1         false        APIService
# selfsubjectreviews                             authentication.k8s.io/v1          false        SelfSubjectReview
# tokenreviews                                   authentication.k8s.io/v1          false        TokenReview
# selfsubjectaccessreviews                       authorization.k8s.io/v1           false        SelfSubjectAccessReview
# selfsubjectrulesreviews                        authorization.k8s.io/v1           false        SelfSubjectRulesReview
# subjectaccessreviews                           authorization.k8s.io/v1           false        SubjectAccessReview
# certificatesigningrequests        csr          certificates.k8s.io/v1            false        CertificateSigningRequest
# flowschemas                                    flowcontrol.apiserver.k8s.io/v1   false        FlowSchema
# prioritylevelconfigurations                    flowcontrol.apiserver.k8s.io/v1   false        PriorityLevelConfiguration
# ingressclasses                                 networking.k8s.io/v1              false        IngressClass
# runtimeclasses                                 node.k8s.io/v1                    false        RuntimeClass
# clusterrolebindings                            rbac.authorization.k8s.io/v1      false        ClusterRoleBinding
# clusterroles                                   rbac.authorization.k8s.io/v1      false        ClusterRole
# priorityclasses                   pc           scheduling.k8s.io/v1              false        PriorityClass
# csidrivers                                     storage.k8s.io/v1                 false        CSIDriver
# csinodes                                       storage.k8s.io/v1                 false        CSINode
# storageclasses                    sc           storage.k8s.io/v1                 false        StorageClass
# volumeattachments                              storage.k8s.io/v1                 false        VolumeAttachment
```

*(or use `--namespaced=true` for all other resources)*

### Selecting Resources From Namespaces

Now if you just created a resource like a `deployment`, `service`, `secret` or `configmap` you can check out the resources specifying the namespace as an argument to the `kubectl get` command like this:

```
kubectl get configmap -n black-space
```

This is really handy and important to make sure you can seperate access rights to resources. If you have several teams that are accessing and altering resources you can give permissions to the team members based on the namespace of the resource.

There is also the tool `kubens` to change the default namespace so that the team members of team `red` do not always have to add the namespace to their `kubectl` commands.

## *Create MongoDB Service*

The example material for this is set up with a namespace called `black-space` run `$> kubectl create namespace black-space` before applying any of the configurations from the folder `03_mongodb`. *(In reality you probably don't want to set up a mongodb in a kubernetes cluster, cause mongodb is quite challenging to separate into partitions, not impossible though since technically you can use chunks, good luck and have fun)*

To make the setup easier in the folder `03_mongodb` there are examples for all of the `.yaml` files that are created in this section. And for the namespace setup you can also use the file `./black-namespace.yaml`. In general whenever you read the word `black` it means, that this is freely chosen and no magic key word.

> It is possible that you will have to change the version `4.4` of the image `docker.io/library/mongo:4.4` to something that suits your system.

1. To bootstrap this deployment, create a *deployment file* (for example black-db.yaml) like this, but do not apply it yet

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: black-db # call this whatever makes sense to you
  namespace: black-space # this has to be created before applying this file
  labels:
    app: db-label # this is how the context is referred to
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db-label # reference to context
  template:
    metadata:
      labels:
        app: db-label # reference to context
    spec:
      containers:
      - name: db-label # reference to context
        image: docker.io/library/mongo:4.4 # the 'image' property corresponds to an actual image [!]
        ports:
        - containerPort: 27017 # this port is actually exposed by this image by default [!]
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          value:
        - name: MONGO_INITDB_ROOT_PASSWORD
          value:
```

2. Before you deploy you create a `Secret` file (for example black-secret.yaml) like this:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: black-secret
  namespace: black-space
type: Opaque
data:
  black-user: bXItYmxhY2s= # this is 'mr-black' base64 encoded
  black-pass: Q2hhbmdlbWUrMTIz # this is 'Changeme+123' base64 encoded
```

3. **[IMPORTANT]:** create the namespace and the secret first running

```
kubectl apply -f ./black-namespace.yaml
# [output] > namespace/black-space created
kubectl apply -f ./black-secret.yaml
# [output] > secret/black-secret created
```

4. Check if the secret and namespace is created running

```
kubectl get namespace
# [output]
# NAME              STATUS   AGE
# black-space       Active   3s
# default           Active   5h33m
# kube-node-lease   Active   5h33m
# kube-public       Active   5h33m
# kube-system       Active   5h33m
kubectl get secret -n black-space
# [output]
# NAME            TYPE     DATA   AGE
# black-secret    Opaque   2      19s
```

5. Now update your deployment file (black-db.yaml) like this:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: black-db # call this whatever makes sense to you
  namespace: black-space
  labels:
    app: db-label # this is how the context is refered to
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db-label # reference to context
  template:
    metadata:
      labels:
        app: db-label # reference to context
    spec:
      automountServiceAccountToken: false # your IDE will recommend you to add this line
      # this is also where you can reference PersistentVolumeClaim in 'volumes' property
      containers:
      - name: db-label # reference to context
        image: docker.io/library/mongo:4.4 # the 'image' property corresponds to an actual image [!]
        resources:
          limits:
            memory: 100Mi # your IDE will recommend you to add this line
        ports:
        - containerPort: 27017 # this port is actually exposed by this image by default [!]
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          valueFrom:
            secretKeyRef:
              name: black-secret
              key: black-user
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: black-secret
              key: black-pass
```

6. Then add a service to the mongodb deployment `black-db` at the bottom of the file like this:

```
# triple dash separates 'kubernetes' yaml definitions
---
apiVersion: v1
kind: Service
metadata:
  name: black-db-service # call this whatever makes sense to you
  namespace: black-space
spec:
  selector:
    app: db-label # reference to context
  ports:
  - protocol: TCP
    port: 27017 # the port exposed to the outer environment
    targetPort: 27017 # the actual port exposed by the pod
```

7. Now apply the files and check that the deployment is correct by running

```
kubectl apply -f ./black-db.yaml
# [output]
# deployment.apps/black-db created
# service/black-db-service created
kubectl get all -n black-space
# [output]
# NAME                            READY   STATUS    RESTARTS   AGE
# pod/black-db-77ffc5c8c6-8bcs7   1/1     Running   0          47s

# NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
# service/black-db-service   ClusterIP   10.111.15.138   <none>        27017/TCP   47s

# NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
# deployment.apps/black-db   1/1     1            1           47s

# NAME                                  DESIRED   CURRENT   READY   AGE
# replicaset.apps/black-db-77ffc5c8c6   1         1         1       47s
```

8. Check that the pod is correctly routed

```
kubectl describe service black-db-service -n black-space
# [output]
# Name:              black-db-service
# Namespace:         black-space
# Labels:            <none>
# Annotations:       <none>
# Selector:          app=db-label
# Type:              ClusterIP
# IP Family Policy:  SingleStack
# IP Families:       IPv4
# IP:                10.111.15.138
# IPs:               10.111.15.138
# Port:              <unset>  27017/TCP
# TargetPort:        27017/TCP
# Endpoints:         10.244.0.23:27017
# Session Affinity:  None
# Events:            <none>
```

9. And that the endpoint has the correct port open (here 10.244.0.23 needs :27017 in output above)

```
kubectl get pod -o wide -n black-space
# [output]
# NAME                        READY   STATUS    RESTARTS   AGE     IP            NODE       NOMINATED NODE   READINESS GATES
# black-db-77ffc5c8c6-8bcs7   1/1     Running   0          2m51s   10.244.0.23   minikube   <none>           <none>
```

## *Create MongoExpress*

1. Make sure you completed all steps from the section *Create MongoDB Service* successfully
2. Create a config file (for example black-express.yaml) for your mongo-express deployment like so $>

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: black-express # call this whatever makes sense to you
  namespace: black-space
  labels:
    app: ui-label
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ui-label
  template:
    metadata:
      labels:
        app: ui-label
    spec:
      automountServiceAccountToken: false
      containers:
      - name: ui-label
        image: docker.io/library/mongo-express
        resources:
          limits:
            memory: 100Mi
        ports:
        - containerPort: 8081
        env:
        - name: ME_CONFIG_MONGODB_ADMINUSERNAME
          valueFrom:
            secretKeyRef:
              name: black-secret
              key: black-user
        - name: ME_CONFIG_MONGODB_ADMINPASSWORD
          valueFrom:
            secretKeyRef:
              name: black-secret
              key: black-pass
        - name: ME_CONFIG_MONGODB_SERVER
          valueFrom:
            configMapKeyRef:
              name: black-configmap
              key: black-db-url

# triple dash separates 'kubernetes' yaml definitions
---
apiVersion: v1
kind: Service
metadata:
  name: black-express-service
  namespace: black-space
spec:
  selector:
    app: ui-label
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8081
    targetPort: 8081 # this is where the container is listening
    nodePort: 30000 # this port has to be inside range [30000-32767]
```

3. Check everything from the chapter *Create MongoDB Service* is up and running

```
kubectl get all -n black-space
# [output]
# NAME                            READY   STATUS    RESTARTS   AGE
# pod/black-db-77ffc5c8c6-xw2tc   1/1     Running   0          23m

# NAME                        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)     AGE
# service/black-db-service    ClusterIP   10.96.70.35   <none>        27017/TCP   23m

# NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
# deployment.apps/black-db    1/1     1            1           23m

# NAME                                   DESIRED   CURRENT   READY   AGE
# replicaset.apps/black-db-77ffc5c8c6    1         1         1       23m
```

4. Create a `ConfigMap` file (for example black-configmap.yml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  # this is referenced in black-express.yaml => ME_CONFIG_MONGODB_SERVER
  name: black-configmap
data:
  # this is referenced in black-express.yaml => ME_CONFIG_MONGODB_SERVER
  black-db-url: black-db-service
```

5. Create the config map **first** by running

```
kubectl apply -f ./black-configmap.yaml
# [output] > configmap/black-configmap created
```

6. Then create the mongo express deployment

```
kubectl apply -f ./black-express.yaml
# [output]
# deployment.apps/black-express created
# service/black-express-service created
```

7. **[IMPORTANT]:** now you expose the express service running
*this will need the podman to have rights to open a network connection*

```
minikube service black-express-service -n black-space
# [output]
# |-------------|-----------------------|-------------|---------------------------|
# |  NAMESPACE  |         NAME          | TARGET PORT |            URL            |
# |-------------|-----------------------|-------------|---------------------------|
# | black-space | black-express-service |        8081 | http://192.168.49.2:30000 |
# |-------------|-----------------------|-------------|---------------------------|
# 🏃  Starting tunnel for service black-express-service.
# |-------------|-----------------------|-------------|------------------------|
# |  NAMESPACE  |         NAME          | TARGET PORT |          URL           |
# |-------------|-----------------------|-------------|------------------------|
# | black-space | black-express-service |             | http://127.0.0.1:42559 |
# |-------------|-----------------------|-------------|------------------------|
# 🎉  Opening service black-space/black-express-service in default browser...
# 👉  http://127.0.0.1:42559
# ❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

8. Log into the mongodb page using the default credentials `admin` and `pass` (make sure to CHANGE these before going to production)

## Clean Up Everything

*Before you continue clean up the environment gracefully running*

```
kubectl delete service black-express-service -n black-space
# [output] > service "black-express-service" deleted
kubectl delete service black-db-service -n black-space
# [output] > service "black-db-service" deleted
kubectl delete deployment black-express -n black-space
# [output] > deployment.apps "black-express" deleted
kubectl delete deployment black-db -n black-space
# [output] > deployment.apps "black-db" deleted
kubectl delete configmap black-configmap -n black-space
# [output] > configmap "black-configmap" deleted
kubectl delete secret black-secret -n black-space
# [output] > secret "black-secret" deleted
kubectl delete namespace black-space
# [output] > namespace "black-space" deleted
```

## Ingress

Ingress is used to redirect incoming requests to internal services. Ingress is where your TLS connection ends and your HTTPS will be interpreted as HTTP. In other words anyone who has access to anything that is behind Ingress will be able to read everything in plain text, this includes passwords, credit cards, user names, mac addresses or in general sensitive data.

It is recommended to salt and hash passwords the first time right in the same component of the browser, that is responsible for the password input tag so that no other component ever touches plain text data, preferrably not even the browser.

### Bare Metal Set Up

To set up the entry point to a kubernetes cluster it is recommendable to use a proxy server that opens the required port for https, rtps and ftps and forwards the requests to the ingress controller. The ingress controller then checks the ingress rules and forwards the requests to the corresponding services.

### *Setup With Minikube*

This section explains how to set up an Ingress Controller to access an NGINX web-service running on the default namespace. This one is a tad complicated so don't expect it to work out of the box it is very likely that you will have to tinker. Since this is a technology topic it is advisable not to tinker too much unless you are absolutely sure you can undo the steps you do.

1. To have some sort of service up and running inside the cluster we create an nginx deployment yaml (black-web.yaml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: black-web-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
    app: black-web
  template:
    metadata:
      labels:
        app: black-web
    spec:
      automountServiceAccountToken: false
      containers:
      - name: black-web
        image: docker.io/library/nginx
        resources:
          limits:
            memory: 100Mi
            ephemeral-storage: "2Gi"
          requests:
            cpu: 0.5
            memory: 100Mi
            ephemeral-storage: "2Gi"
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: black-web-service # this will be referenced in the ingress rule
spec:
  selector:
    app: black-web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

*(most of these values were explained in previous sections)*

2. Then we setup the deployment running

```
kubectl apply -f black-web.yaml
# [output]
# deployment.apps/black-web-deployment created
# service/black-web-service created
```

3. Enable minikube addon `ingress` by running

```
minikube addons enable ingress
# [output]
#  💡  ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
# You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
#     ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.10.1
#     ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.1
#     ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.1
#  🔎  Verifying ingress addon...
#  🌟  The 'ingress' addon is enabled
```

4. Enable minikube addon `ingress-dns` by running

```
minikube addons enable ingress-dns
# [output]
#  💡  ingress-dns is an addon maintained by minikube. For any concerns contact minikube on GitHub.
# You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
#     ▪ Using image gcr.io/k8s-minikube/minikube-ingress-dns:0.0.2
#  🌟  The 'ingress-dns' addon is enabled
```

5. To show the automatically created ingress controller `$> kubectl get namespace` check if `ingress-nginx` namespace is there then run

```
kubectl get pod -n ingress-nginx
# [output]
# NAME                                        READY   STATUS      RESTARTS   AGE
# ingress-nginx-admission-create-7cd8b        0/1     Completed   0          116s
# ingress-nginx-admission-patch-mk4xn         0/1     Completed   2          116s
# ingress-nginx-controller-768f948f8f-79g82   1/1     Running     0          116s
```

6. To create the dashboard namespace run

```
minikube dashboard
# [output]
#  💡  Enabling dashboard ...
#     ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
#     ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
#  💡  Some dashboard features require the metrics-server addon. To enable all features please run:

#         minikube addons enable metrics-server

#  🤔  Verifying dashboard health ...
#  🚀  Launching proxy ...
#  🤔  Verifying proxy health ...
#  🎉  Opening http://127.0.0.1:40027/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your
default browser...
#  👉  http://127.0.0.1:40027/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
```

*(after this you can open the dashboard entering the url on the last line)*

7. Now run the namespace command to see `ingress-nginx`

```
kubectl get namespace
# [output]
# NAME                   STATUS   AGE
# default                Active   7h36m
# ingress-nginx          Active   5m29s
# kube-node-lease        Active   7h36m
# kube-public            Active   7h36m
# kube-system            Active   7h36m
# kubernetes-dashboard   Active   3m
```

8. To see all pods, services, deployments and replicasets in the kubernetes-dashboard namespace run

```
kubectl get all -n kubernetes-dashboard
# [output]
# NAME                                         READY   STATUS    RESTARTS   AGE
# pod/dashboard-metrics-scraper-b5fc48f67-clvmv   1/1     Running   0          3m33s
# pod/kubernetes-dashboard-779776cb65-s754c       1/1     Running   0          3m33s

# NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
# service/dashboard-metrics-scraper   ClusterIP   10.107.10.173   <none>        8000/TCP   3m33s
# service/kubernetes-dashboard        ClusterIP   10.108.231.41   <none>        80/TCP     3m33s

# NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
# deployment.apps/dashboard-metrics-scraper   1/1     1            1           3m33s
# deployment.apps/kubernetes-dashboard        1/1     1            1           3m33s

# NAME                                                   DESIRED   CURRENT   READY   AGE
# replicaset.apps/dashboard-metrics-scraper-b5fc48f67   1         1         1       3m33s
# replicaset.apps/kubernetes-dashboard-779776cb65       1         1         1       3m33s
```

9. Now create the actual ingress rule in a yaml file like this

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: black-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: black-out.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: black-web-service # this is a reference to the running service
            port:
              number: 80
```

10. Apply this rule and check it got created

```
kubectl apply -f black-ingress.yaml
# [output] > ingress.networking.k8s.io/black-ingress created
kubectl get ingress
# [output]
# NAME            CLASS   HOSTS          ADDRESS        PORTS   AGE
# black-ingress   nginx   black-out.io   192.168.49.2   80      2m40s
```

11. Now you have to update the host file of your WSL2 environment

```
minikube ip
# [output] > 192.168.49.2
sudo nano /etc/hosts
# [output] > opens file in nano
# enter following line below the other definitions
192.168.49.2    black-out.io
```

*(the content of your file should now look something like this)*

```
# This file was automatically generated by WSL. To stop automatic generation of this file, add the following entry to
/etc/wsl.conf:
# [network]
# generateHosts = false
127.0.0.1       localhost
127.0.1.1       YOUR-HOST.domain.your-domain.com      YOUR-HOST

# [!] this is the additional line, the rest is already there
192.168.49.2    black-out.io

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

12. Now you need to do the same on the hosts file of the Windows 11 Pro system

```
# open a PowerShell session with admin permissions and go to the systems directory
cd C:\Windows\System32\drivers\etc\
# open the host file using whatever text editor you like (in this case nano is installed on the system)
nano hosts
# [output] > opens file in nano or complains that nano is not installed
```

*(the content of your file should now look like this)*

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

# [!] this is the additional line, the rest is already there
192.168.49.2    black-out.io

# End of section
```

13. Create a static route from the Windows 11 Pro system to the WSL2 sub system like so

```
# within WSL2 on your bash session find out your IP
ip -4 addr show eth0 | grep -oP '(?<=inet\s)\d+(\.\d+){3}'
# [output] > 172.28.30.69
route add 192.168.49.0 mask 255.255.255.0 172.28.30.69 metric 1
```

14. To clean up everything run following commands in sequencial order

```
kubectl delete -f black-web.yaml
# [output]
# deployment.apps "black-web-deployment" deleted
# service "black-web-service" deleted
kubectl delete -f black-ingress.yaml
# [output] > ingress.networking.k8s.io "black-ingress" deleted
minikube stop
# [output]
# ✋  Stopping node "minikube"  ...
# 🔘  Powering off "minikube" via SSH ...
# ✋  Stopping node "minikube"  ...
# 🔘  Powering off "minikube" via SSH ...
# ✋  Stopping node "minikube"  ...
# 🔘  Powering off "minikube" via SSH ...
# ✋  Stopping node "minikube"  ...
# 🔘  Powering off "minikube" via SSH ...
# ✋  Stopping node "minikube"  ...
# 🔘  1 node stopped.
minikube delete --all
# 🔥  Deleting "minikube" in podman ...
# 🔥  Removing /home/woobie/.minikube/machines/minikube ...
# 💀  Removed all traces of the "minikube" cluster.
# 🔥  Successfully deleted all profiles
podman machine stop
# [output]
# Waiting for VM to exit...
# Machine "podman-machine-default" stopped successfully
podman machine rm -f
# [output] > none
sudo shutdown
# [output] > Shutdown scheduled for Thu 2024-08-08 15:00:30 CEST, use 'shutdown -c' to cancel.
```

15. Delete the static route from your Windows 11 Pro host system again

```
route delete 192.168.49.0
route print all
# [output] > no route to 192.168.49.0
```

### *Start Everything Back Up*

*to restart the entire thing you can enter following commands from a powershell*

```
wsl
# opens a bash session in your WSL2
sudo chmod 777 -R /dev/kvm
sudo mount --make-rshared /
podman machine init --cpus 6 --disk-size 256 --memory 8192
podman machine set --rootful
podman machine start
minikube start --driver=podman --container-runtime=cri-o --network=host
minikube addons enable ingress
minikube addons enable ingress-dns
kubectl apply -f black-web.yaml
kubectl apply -f black-ingress.yaml
ip -4 addr show eth0 | grep -oP '(?<=inet\s)\d+(\.\d+){3}'
# [output] > 172.28.30.69
route add 192.168.49.0 mask 255.255.255.0 172.28.30.69 metric 1
```

After that you can open a browser and type `http:black-out.io` into the URL-bar to see the nginx-welcome page.

## *Troubleshooting*

To check what nodes are running (in this case it is only "minikube") and what pods are running on which node use this:

```
# you may need to install 'jq' first 'sudo apt-get install jq -y'
kubectl get pod --all-namespaces -o json | jq '.items[] | .spec.nodeName + " " + .metadata.name'
# [output]
# "minikube black-web-deployment-7454f6586-l5wbs"
# "minikube ingress-nginx-admission-create-7cd8b"
# "minikube ingress-nginx-admission-patch-mk4xn"
# "minikube ingress-nginx-controller-768f948f8f-79g82"
# "minikube coredns-7db6d8ff4d-h2g2f"
# "minikube etcd-minikube"
# "minikube kindnet-zvw7n"
# "minikube kube-apiserver-minikube"
# "minikube kube-controller-manager-minikube"
# "minikube kube-ingress-dns-minikube"
# "minikube kube-proxy-ckrnp"
# "minikube kube-scheduler-minikube"
# "minikube storage-provisioner"
# "minikube dashboard-metrics-scraper-b5fc48f67-clvmv"
# "minikube kubernetes-dashboard-779776cb65-s754c"
kubectl describe node minikube
# [ouptut] > all relevant information of the node in this case 'minikube'
```

If a pod refuses to get deleted you can run:

```
kubectl delete pod $YOUR_PODNAME --grace-period=0 --force -n $YOUR_NAMESPACE
```

But be careful, this is a last resort. You can introduce memory leaks and all sorts of problems using this method.