# Practice M1: Core Concepts

For the purpose of this practice, we will assume that we are working on a machine with either a Windows 10/11 or any recent Linux distribution and there is a local virtualization solution (like VirtualBox, Hyper-V, VMware Workstation, etc.) installed

## Part 0: Warm up with Containers

Execute the following steps only if you need to refresh your container-related skills

For this part we will assume that we have access to a **Docker** instance (either in a locally installed, VM-based, or elsewhere)

In this preparation part we will warm up with containers in **Docker** by doing the following:

* test the process-container relation
* background services (or detached continuously running containers)
* port redirection
* data exchange with the host
* layered filesystem

Let's start a simple **Alpine**-based container

**docker container run --name alp1 -it alpine sh**

Explore a bit (try commands like **ls**, **ps**, **uname**, **hostname**, etc. and then execute the **exit** command

**exit**

Now check if there are any containers running with

**docker container ls**

None. Let's change the command to

**docker container ls -a**

Aha, here is our container. Let's remove it

**docker container rm alp1**

And start a new one but this time with

**docker container run --name alp1 -it --rm alpine sh**

Explore a bit and once done, hit **Ctrl+P Ctrl+Q** to close the session

Now check if there are any containers running with

**docker container ls**

Yes, it is there and running

To return back, execute

**docker container attach alp1**

Now, let's close the session and exit the container

**exit**

Let's start a container out of the official **NGINX** image

**docker container run --name web1 -d nginx**

Check what containers are running

**docker container ls**

Try to access the web page in the running container

**curl http://localhost**

No success

Let's stop and remove the container

**docker container rm --force web1**

Now, start a new one but with port redirection (you may need to adjust the rule - ***host-port:container-port***)

**docker run --name web1 -d -p 80:80 nginx**

Try to access the web page in the running container

**curl http://localhost**

This time it is working as expected

Let's stop and remove the container

**docker container rm --force web1**

What if we want to use a custom web page?

First option is to mount or specify one during the container creation process

Let's create a folder with a custom **index.html** file in it

**mkdir web**

**echo 'My cutom page :)' > web/index.html**

Now, run a container with it

**docker run --name web1 -d -p 80:80 -v $(pwd)/web:/usr/share/nginx/html:ro nginx**

Try to access the web page in the running container

**curl http://localhost**

Again, it is working as expected

Let's stop and remove the container

**docker container rm --force web1**

Now, let's create our own image and test the layered filesystem

Create a **Dockerfile** file with the following content

**FROM nginx**

**COPY web/index.html /usr/share/nginx/html**

Create the image with

**docker image build -t my-nginx:v1 .**

List the images

**docker image ls**

Now, run a container based on our image

**docker run --name web1 -d -p 80:80 my-nginx:v1**

Try to access the web page in the running container

**curl http://localhost**

We should see the new (the one made by us) default index page

Open a session to the container

**docker container exec -it web1 bash**

And change the **/usr/share/nginx/html/index.html** file

**echo 'Another page :)' > /usr/share/nginx/html/index.html**

Close the session

**exit**

Try to access the web page in the running container

**curl http://localhost**

Now, it should display the changed version

Check the changes that happened to the container's filesystem

**docker container diff web1**

Let's stop and remove the container

**docker container rm --force web1**

We warmed up and are ready to move forward 😉

## Part 1: Introduction to Kubernetes

In this part we will focus on the creation of simple yet working environment to start our journey in the world of **Kubernetes**

### Install minikube

Installation of **minikube** is simple in every operating system

#### Linux

For **Linux** distributions we must execute the following steps

Open a terminal session and download the binary

**curl -LO** [**https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64**](https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64)

“Install” the binary in a folder that is part of the **PATH** environment variable, for example

**sudo install minikube-linux-amd64 /usr/local/bin/minikube**

#### Windows

For **Windows**, we must download the installer from here:

<https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe>

Then follow the usual steps – double-click and then Next, Next, Next 😉

#### Other OSes

For other OSes or options check here: <https://minikube.sigs.k8s.io/docs/start/>

### Work with minikube

Now that we have it installed, let’s check if the binary is reachable

**minikube version**

Let’s see the list of available commands

**minikube**

We can check the status of the **minikube** cluster

**minikube status**

Currently, we do not have one (yet)

Check which driver (**docker**, **virtualbox**, etc.) will be used to start the cluster if we decide to do it

**minikube start --dry-run=true**

Start the cluster with the driver of your choice

**minikube start --driver=<driver-name>**

*If you are using* ***VMware Workstation*** *and the above doesn't work, make sure that the folder in which* ***vmrun.exe*** *resides is included in the* ***PATH*** *variable. You can always check with* ***echo %PATH%***

Should we want to fine-tune the cluster creation (for example, bigger disk, more CPU, …) we must check what are the available options including those specific to our hypervisor

**minikube start --help**

Now, that we have our single-node **Kubernetes** cluster, we can establish an **SSH** session to the virtual machine

**minikube ssh**

Let’s check the default user

**id**

Then, check the installed version of **Docker**

**docker version**

Check if there are any images available locally

**docker image ls**

Check if there are any containers running

**docker container ls**

We are done here for now. Close the **SSH** session

**exit**

### Dashboard

The **Kubernetes** dashboard is implemented as an addon in **minikube**

We can check the list of available addons and their status with

**minikube addons list**

We can see that by default it is not enabled

This can be done like this (**skip it**)

**minikube addons enable dashboard**

If we go this way, we will see that we must enable one more addon – **metrics-server**

**minikube addons enable metrics-server**

Instead, we will do it with the following command

**minikube dashboard**

It will enable the dashboard addons and then it will start a proxy and open a browser which will navigate us to the start page of the dashboard

Explore a bit. Once done, return on the terminal and hit **Ctrl+C** to stop the proxy

### Install kubectl

We should have a way to control our cluster from the command line. For this we will need the **kubectl** binary

#### Linux

For **Linux** distributions we must execute the following steps

Open a terminal session and download the latest binary

**curl -LO** [**https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl**](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

Then “install” it to a folder that is part of the **PATH** environment variable

**sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl**

#### Windows

For **Windows** we must download the binary from here:

**https://dl.k8s.io/release/v1.22.2/bin/windows/amd64/kubectl.exe**

We may need to change the version. The latest version can be seen here: <https://dl.k8s.io/release/stable.txt>

Then copy the file in a folder that is part of the **PATH** environment variable

#### Other OSes

For other OSes or options check here: <https://kubernetes.io/docs/tasks/tools/install-kubectl/>

### Getting to know kubectl

First, we must check if the binary is reachable

**kubectl version --client**

Then list all available commands

**kubectl**

We can retrieve information for the cluster that is configured currently (this should be the **minikube**)

**kubectl cluster-info**

As we do not have many things to see yet, we can ask for the nodes of our cluster (yes, we know it is just one)

**kubectl get nodes**

Should we want more information, we can extend the command to

**kubectl get nodes -o wide**

Let’s stop for now and do a short break. Then we will continue with our journey

## Part 2:

### Objects Exploration

Now, that we know for the existence of objects and their purpose, let’s see a list of all supported resources

**kubectl api-resources**

We can see that some of them have a long and short name

Let’s list all supported **API** versions

**kubectl api-versions**

Get information about the **POD** resource

**kubectl explain pod**

Get information about the **SPEC** section of the **POD** resource

**kubectl explain pod.spec**

Filter the required fields in the **SPEC** section of the **POD** resource

*For Linux/macOS we can use*

***kubectl explain pod.spec | grep required***

*And on Windows this*

***kubectl explain pod.spec | find.exe /I `"required`"***

We can see that there is just one required field

So, let’s get information about the **CONTAINERS** portion of the **SPEC** section of the **POD** resource

**kubectl explain pod.spec.containers**

Wow, plenty of fields. Again, we can check which are required if we want

### Namespaces

We know now that the cluster can be divided on logically separated clusters by defining namespaces

We can do this imperatively with command

**kubectl create namespace ns1**

And then check if it is there

**kubectl get namespaces**

Let’s check what else we can create with this command

**kubectl create --help**

As we can see, we can create plenty of objects

We can even simulate the creation process and see the actual **YAML** code

**kubectl create namespace ns2 --dry-run=client --output=yaml**

This can be stored and then reused. Let’s do it

**kubectl create namespace ns2 --dry-run=client --output=yaml > ns2.yaml**

Now, use the file to create the resource imperatively again

**kubectl create -f ns2.yaml**

And list the available namespaces again

**kubectl get namespaces**

We can shorten **namespaces** to **ns**

Delete the second namespace either with

**kubectl delete namespaces ns2**

Or with

**kubectl delete -f ns2.yaml**

Should you want, continue experimenting but don’t forget to leave one extra namespace (for example ns1)

### Pods

Okay, let’s retrieve a list of all pods (in the **default** namespace)

**kubectl get pods**

Nothing to see here, yet

Get the pods in all namespaces

**kubectl get pods --all-namespaces**

Some of the options have a long and short version. For example, the above command can be shortened to

**kubectl get pods -A**

Now, let’s try an imperative creation of a pod with command

Create a **NGINX** based pod

**kubectl run nginx-pod --image nginx**

Check the result

**kubectl get pods**

If we, do it immediately, we will notice that the pod is in **ContainerCreating** status. We should wait several seconds and try again. Now it should be in a **Running** status

Remove the pod

**kubectl delete pod nginx-pod**

Now, create it again but this time in the **ns1** namespace

**kubectl run nginx-pod --image nginx -n ns1**

Wait where did this **-n** came from? Let’s check the **kubectl run** options

**kubectl run --help**

It appears that there isn’t such option. Yes, everything is fine, we have the so-called global options, which we can see with

**kubectl options**

Aha, here it is

Let’s check if it indeed was created there with

**kubectl get pods -A**

As we did earlier, we can ask **kubectl run** to create an **YAML** configuration for us (perhaps you noticed the same options in its help as with the **create** command)

So, let’s do it

**kubectl run nginx-pod-yaml --image nginx -n ns1 --dry-run=client -o=yaml**

Two things to note here – we can use create and run to generate **YAML** for us; and we can mix short and long options in one command (should we do it is another story)

Before we continue, let’s delete the namespace to prove that all objects belonging to it will be deleted

Delete the **ns1** namespace

**kubectl delete namespace ns1**

Check again for all pods in all namespaces

**kubectl get pods -A**

They both (the namespace and the pod) are gone

Let’s test once more the imperative creation with configuration file

First, check the contents of the file

*For Linux we can use this*

***vi demo-files/1-appa-pod.yml***

*And under Windows we can use the VSCode editor*

***code demo-files/1-appa-pod.yml***

Create the pod in an imperative fashion but using a file

**kubectl create -f demo-files/1-appa-pod.yml**

And check the result

**kubectl get pods**

Display detailed information about the pod

**kubectl describe pod appa-pod**

Explore the information. There is plenty to learn here for a pod

Check the contents of the new version of the file

*For Linux we can use this*

***vi demo-files/2-appa-pod-ext.yml***

*And under Windows we can use the VSCode editor*

***code demo-files/2-appa-pod-ext.yml***

We can compare the initial configuration file with its extended version

*For Linux we can use this*

***vimdiff demo-files/1-appa-pod.yml demo-files/2-appa-pod-ext.yml***

*And under Windows we can use PowerShell*

**Compare-Object (Get-Content -Path demo-files/1-appa-pod.yml) -DifferenceObject (Get-Content -Path demo-files/2-appa-pod-ext.yml) -IncludeEqual**

Apply the changes coming from the extended file

**kubectl apply -f demo-files/2-appa-pod-ext.yml**

Note the warning (it is because of the imperative creation earlier)

Display detailed information about the pod

**kubectl describe pod appa-pod**

Explore the labels section

### Labels and Annotations

Now, let’s do a few more experiments with labels and annotations and then we will continue with the main course

Create two pods with the following commands

**kubectl run nginx-1 --image=nginx --labels="image=nginx,ver=v1" --annotations="created-by=user1"**

**kubectl run nginx-2 --image=nginx --labels="image=nginx,ver=v2" --annotations="created-by=user1"**

List the pods in the **default** namespace

**kubectl get pods**

We can ask even for more information with

**kubectl get pods -o wide**

Let’s change the previous command to show the labels as well

**kubectl get pods --show-labels**

We can ask the labels to be presented as columns

**kubectl get pods -L image,ver**

Now, filter the pods by label. For example, ask for all pods for which the **ver** is set to **v1**

**kubectl get pods -l ver=v1**

Or all with **image** label equal to **nginx**

**kubectl get pods -l image=nginx**

Okay, but where are the annotations? We can see them (together with the labels) in the output of the describe command

**kubectl describe pod nginx-1**

Can we change a label? Sure, we can. Let’s change the **ver** label of the **nginx-2** pod to **v1**

Execute the following

**kubectl label --overwrite pods nginx-2 ver=v1**

If we ask again for the list of pods, we will see that there is a change

**kubectl get pods --show-labels**

Let’s use the labels to delete the two extra pods

**kubectl delete pods -l image=nginx**

### Services

Let’s check what we have in terms of running pods

**kubectl get pods**

Okay, it should be only our **appa-pod**

How can we access or consume the service that is provided by the container inside the pod?

We can expose the service running on the pod and make it reachable on the IP address of our single-node cluster

**kubectl expose pod appa-pod --name=appa-svc --target-port=80 --type=NodePort**

Display information about the service

**kubectl get svc appa-svc**

Show detailed information about the service

**kubectl describe svc appa-svc**

Check how the service can be reached

**minikube service list**

Copy the **appa-svc URL** and paste it to a browser tab. You should see a working "application"

Let’s remove the service

**kubectl delete svc appa-svc**

Explore the configuration file that will create a similar service

*For Linux we can use this*

***vi demo-files/3-appa-svc.yml***

*And under Windows we can use*

***code demo-files/3-appa-svc.yml***

Create the service in a declarative manner

**kubectl apply -f demo-files/3-appa-svc.yml**

Display detailed information about the service

**kubectl describe svc appa-svc**

Note the **Endpoints** position. It contains a reference to the pod

Create a copy of the file **demo-files/2-appa-pod-ext.yml** and change just the pod name to **appa-pod-1**

Then save it and create the pod

Alternatively, you can execute the following imperative command to save some time

**kubectl run appa-pod-1 --image=shekeriev/k8s-appa:v1 --labels="app=appa,ver=v1"**

Ask again for detailed information about the service

**kubectl describe svc appa-svc**

Now, there are two pods served by the service

Ask again for the available services

**minikube service list**

Copy the **appa-svc URL** and paste it to a browser tab. The "application" should be reachable

Note that the port is different. This time we set it to a fixed value

Refresh a few times. You should notice that requests are served by different pods

Remove the first pod

**kubectl delete pod appa-pod**

Refresh the open browser tab. You will notice that the "application" is still working

Delete the other one as well

**kubectl delete pod appa-pod-1**

Refresh the open browser tab. Now, the "application" should not be reachable

Show detailed information about the service

**kubectl describe svc appa-svc**

Note the **Endpoints** position. It is empty

## Part 3:

As we will see both replication controllers and replica sets look and behave similar

### Replication Controllers

Explore the configuration file to create a sample replication controller

*For Linux we can use this*

***vi demo-files/4-rc.yml***

*And under Windows we can use*

***code demo-files/4-rc.yml***

Create the replication controller

**kubectl apply -f demo-files/4-rc.yml**

Show information about the replication controllers

**kubectl get rc**

And with more details

**kubectl get rc -o wide**

We can describe the replication controller as we did with the objects so far

**kubectl describe rc appa-rc**

And now, describe the service

**kubectl describe services appa-svc**

Check how the service can be reached

**minikube service list**

Copy the **appa-svc URL** and paste it to a browser tab. You should see a working "application"

Now, let’s scale up the replication controller

**kubectl scale --replicas=5 rc/appa-rc**

List the replication controllers

**kubectl get rc**

And check the service again

**kubectl describe services appa-svc**

Not all endpoints are seen. Try **describe** but with the endpoint this time

**kubectl describe endpoints appa-svc**

Refresh a few times the open browser tab. You should notice that requests are served by different pods

Let’s scale down to 2 replicas

**kubectl scale --replicas=2 rc/appa-rc**

List the replication controllers

**kubectl get rc**

Finally, delete the replication controller

**kubectl delete -f demo-files/4-rc.yml**

### Replica Sets

Explore the configuration file to create a sample replica set

*For Linux we can use this*

***vi demo-files/5-rs.yml***

*And under Windows we can use*

***code demo-files/5-rs.yml***

Create the replica set

**kubectl apply -f demo-files/5-rs.yml**

Show information about the replica sets

**kubectl get rs**

And with more details

**kubectl get rs -o wide**

We can describe the replica set as we did with the objects so far

**kubectl describe rs appa-rs**

And now, describe the service

**kubectl describe services appa-svc**

Check how the service can be reached

**minikube service list**

Copy the **appa-svc URL** and paste it to a browser tab. You should see a working "application"

Now, let’s scale up the replica set

**kubectl scale --replicas=5 rs/appa-rs**

List the replica sets

**kubectl get rs**

And check the service again

**kubectl describe services appa-svc**

Not all endpoints are seen. Try **describe** but with the endpoint this time

**kubectl describe endpoints appa-svc**

Refresh a few times the open browser tab. You should notice that requests are served by different pods

Let’s scale down to 2 replicas

**kubectl scale --replicas=2 rs/appa-rs**

List the replica sets

**kubectl get rs**

Finally, delete the replica set

**kubectl delete -f demo-files/5-rs.yml**

### Deployments

Let’s create a deployment with two pod replicas

**kubectl create deployment appa-deploy --image=shekeriev/k8s-appa:v1 --replicas=2 --port=80**

List deployment objects

**kubectl get deployments**

Show detailed information about our deployment

**kubectl describe deployment appa-deploy**

Scale up the deployment to ten pod replicas

**kubectl scale deployment appa-deploy --replicas=10**

Watch how the pod replicas are being created

**kubectl get pods -w**

Press **Crl+C** to stop the pods monitoring process

Check if all ten pods are there

**kubectl get pods**

Remove the deployment together with the replicated pods

**kubectl delete deployment appa-deploy**

Check what is happening with the pods

**kubectl get pods**

They are terminating

Now, let’s try the declarative approach

Explore the configuration file that will be used to create a deployment

*For Linux we can use this*

***vi demo-files/6-appa-deploy-v1.yml***

*And under Windows we can use*

***code demo-files/6-appa-deploy-v1.yml***

Create the deployment in a declarative manner

**kubectl apply -f demo-files/6-appa-deploy-v1.yml**

Watch while the pods are being created

**kubectl get pods -w**

Press **Crl+C** to stop the pods monitoring process

Ask for deployment status

**kubectl get deployments**

Ask for detailed deployment status

**kubectl get deployments -o wide**

Note the **SELECTOR** column content

Refresh the open browser tab a few times and pay attention where the "application" is running

Now, we will deploy a "newer" version

Check the contents of the new version of the file

*For Linux we can use this*

***vi demo-files/7-appa-deploy-v2.yml***

*And under Windows we can use*

***code demo-files/7-appa-deploy-v2.yml***

Compare the two versions of the deployment

*Under Linux use this*

***vimdiff demo-files/6-appa-deploy-v1.yml demo-files/7-appa-deploy-v2.yml***

*Under Windows use this*

***Compare-Object (Get-Content -Path demo-files/6-appa-deploy-v1.yml) -DifferenceObject (Get-Content -Path demo-files/7-appa-deploy-v2.yml) -IncludeEqual***

Retrieve detailed information about the current deployment

**kubectl describe deployment appa-deploy**

List current replica sets

**kubectl get rs**

Retrieve detailed information about the only replica set (if there were many, we should have specified the name as well)

**kubectl describe rs**

Apply the newer deployment configuration but record the changes

**kubectl apply -f demo-files/7-appa-deploy-v2.yml --record**

Watch the deployment rollout. It is done one pod at a time

**kubectl rollout status deployment appa-deploy**

Refresh the open browser tab a few times. You will notice that some of the requests will be served by the old version of the "application" and others by the new one

Retrieve the history of the deployment

**kubectl rollout history deployment appa-deploy**

Undo the latest deployment and return the previous version of the "application"

**kubectl rollout undo deployment appa-deploy --to-revision=1**

Watch the rollback process

**kubectl rollout status deployment appa-deploy**

Refresh the open browser tab. You will notice that some of the requests will be served by the old version of the "application" and others by the new one

Retrieve the history of the deployment

**kubectl rollout history deployment appa-deploy**

### Clean up

Remove the deployment together with the replica set and all the pods

**kubectl delete deployment appa-deploy**

Remove the service as well

**kubectl delete service appa-svc**

Check that there are not any unwanted resources left

**kubectl get all --all-namespaces**

### Extra: Services and DNS

Let’s deploy a two-pod application

Deploy the producer pod + service (backend part)

**kubectl apply -f demo-files-extra/producer-pod.yml**

**kubectl apply -f demo-files-extra/producer-svc.yml**

Let's spin another one to act as an observer

**kubectl apply -f demo-files-extra/observer-pod.yml**

Connect to it

**kubectl exec -it observer-pod -- sh**

And install the **curl** command

**apk add curl**

Now, check if the service is accessible by name (**producer**)

**curl http://producer:5000**

Now, try the other names (service + namespace & **FQDN**) of the service

**curl http://producer.default:5000**

**curl http://producer.default.svc.cluster.local:5000**

Notice the name of the pod

Exit the observer session

**exit**

Delete the pod

**kubectl delete -f demo-files-extra/producer-pod.yml**

And spin up a deployment with 3 replicas

**kubectl apply -f demo-files-extra/producer-deployment.yml**

Then check the pods

**kubectl get pods**

Open again a session to the "observer"

**kubectl exec -it observer-pod -- sh**

Now, check if the service is accessible by name (producer)

**curl http://producer:5000**

Re-execute a few times and pay attention to the pod name

Close the session

**exit**

Deploy the consumer pod + service (frontend part)

**kubectl apply -f demo-files-extra/consumer-pod.yml**

**kubectl apply -f demo-files-extra/consumer-svc.yml**

Check the pods and services

**kubectl get pods,services**

Get the service URL with

**minikube service list**

Open a browser tab to the URL and refresh a few times and pay attention to the IDs on top and bottom of the page

It is time to delete the consumer pod

**kubectl delete -f demo-files-extra/consumer-pod.yml**

And create the consumer deployment

**kubectl apply -f demo-files-extra/consumer-deployment.yml**

Get the service URL with

**minikube service list**

Open a browser tab to the URL and refresh a few times and pay attention to the IDs on top and bottom of the page

### Clean up 2

Remove all created resources

**kubectl delete -f demo-files-extra/observer-pod.yml**

**kubectl delete -f demo-files-extra/producer-deployment.yml**

**kubectl delete -f demo-files-extra/producer-svc.yml**

**kubectl delete -f demo-files-extra/consumer-deployment.yml**

**kubectl delete -f demo-files-extra/consumer-svc.yml**

Check that the deletion succeeded

**kubectl get all**

### Clean up 3

To delete the **minikube**, we must execute

**minikube delete**