# Practice M8: Exam Preparation

## Exam Plot

Should you want to reproduce the environment, you will have to prepare the following set of machines:

* **Exam station** is with **kubectl** + **helm** + **kustomize** installed
* **Mars cluster** – two machines – working cluster with **Flannel** as a pod network plugin and **HAProxy** as ingress controller
* **Jupiter cluster** – three machines – working cluster with **WeaveNet** as a pod network plugin
* **Venus cluster** – two machines – **broken** cluster with **Antrea** as a pod network plugin (should be installed as prat of the solution)

All clusters are **Kubernetes 1.21.6** and all their nodes are with **2 vCPU / 2 GB RAM / 32 GB Disk / Debian 10**. The exam station is with **1 vCPU / 1 GB RAM**

There is the **exam** user which has **sudo** privileges on all machines and can authenticate to each one of them via SSH

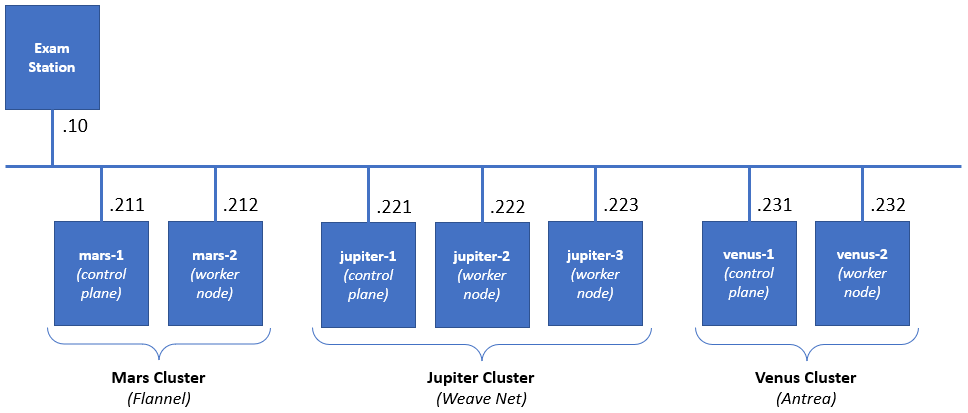
If you are constrained by the available resources, you can spin up the clusters one by one, as the tasks are grouped by cluster. You can even omit the exam station and use your host to control and work with the clusters

Check the published archive for the setup files (folder **1-setup**). There is a separate folder (**2-solution**) with the final versions of the files (you are expected to create them as part of the solution) but try to not use it, at least not before you tried to solve it yourself

## Sample Exam

### Infrastructure

You will have to accomplish a set of tasks in the following infrastructure



### Rules

Be sure to **follow strictly** the **naming** **conventions** specified in the checklist

Tasks execution order should not be derived from the order in which they are listed below. Please note that there are tasks that depend on the successful completion of one or more other tasks

Usually, all steps could be achieved by following different paths and using different tools. In the end, not the means, but the **results** are being **measured**, **except stated otherwise**

### Tasks checklist

#### Mars Cluster [18 pts]

* (T101 / 3 pts) There is the **animals** namespace. There are two pairs of pod and service. You are expected to create an ingress resource (using the available ingress controller and class) that
  + Will be in the same namespace and named **pets-ingress**
  + Will serve the **pets.lab** host
  + Path **/cat** to be redirected to **cat-svc** service and **/dog** to the **dog-svc**
  + Store it in **/files/mars/t101-out.yaml** and be sure to deploy it
* (T102 / 3 pts) Explore the **tiger** namespace. There is a pod that is not in running state, but it should be. Its manifest is **/files/mars/t102-in.yaml**. Your mission, should you accept it, is to
  + Correct the issue(s) and reflect this in a new manifest **/files/mars/t102-out.yaml**
  + Make sure that the pod is in running state and doesn’t restart periodically because of a probe
* (T103 / 4 pts) Templating is good and necessary technique. We have a simple manifest **(/files/mars/t103-in.yaml**) which we want to be able to easily deploy in production (**blue**) and in test (**green**). Using the **kustomize** application, you must prepare a set of folders and files in the **/files/mars/t103** folder that allows
  + Base (without any changes) deployment and deployment to both environments
  + The **blue** deployment should increase the **replicas** to **3**, use the **blue** tag, and runs on port **31103**
  + The **green** deployment should use the **green** image tag and runs on port **32103**
  + Make sure that blue and green are deployed (but not the base)
* (T104 / 2 pts) Explore the **cherry** namespace. There is a deployment that is failing. Your mission is to
  + Find the reason for this and write it down *(****type of the object:name of the object****, for example* ***limit:banana****)* in the **/files/mars/t104-reason.txt** file
  + Correct the situation by changing the offending parameter of the deployment to comply
* (T105 / 2 pts) Explore the pod manifest in **/files/mars/t105-in.yaml** file and
  + Create a new one (**/files/mars/t105-out.yaml**) that wraps the pod template in a **CronJob** named **five-job**
  + Set it to run **every 5 minutes** and deploy it
* (T106 / 4 pts) There is the **fortress** namespace. It is empty. Your mission is to
  + Create a **ServiceAccount** named **observer**
  + Create a **Role** named **looknotouch** that allows only **get** on **pods**
  + Create a **RoleBinding** named **looknotouch** that binds the role to the service account
  + Modify the pod manifest **/files/mars/t106-in.yaml** to run the pod with the observer service account, store the new version in **/files/mars/t106-out.yaml** and deploy it

#### Jupiter Cluster [25 pts]

* (T201 / 4 pts) There are three namespaces – **apple**, **orange**, and **apricot**. In all three, there is a pair of a pod and service. There aren’t any restrictions. You should correct this:
  + Add a network policy that will limit the **ingress access** to the pods in the **apple** namespace to connections, coming only from pods in the **orange** namespace
* (T202 / 5 pts) We all know that using **Helm** charts is both fun and easy. So, let’s spin up one chart
  + Use the **artifacthub.io** and find the **Apache HTTP** chart provided by **Bitnami** and **add the repository**
  + Then **install** the chart as **exam-httpd** release in the **kiwi** namespace (create it if not existing)
  + Make sure that using the chart’s parameters it is set to use a service of type **NodePort** and to listen for HTTP request on port **32202**
  + Create a **ConfigMap** named **exam-httpd-cm** in the same namespace that contains an **index.html** file with the following text **Helm+Kubernetes=Fun** and attach it to the release
* (T203 / 4 pts) There is a pod in the **cucumber** namespace that is consuming a secret in the same namespace. You are expected to:
  + Find the unencoded value of the secret and save it as **/files/jupiter/t203-secret.txt**
  + Then change the secret to **Cucumbers are green**
* (T204 / 2 pts) Add a new label (**exam**) to both worker nodes (**jupiter-2** and **jupiter-3**):
  + Set it to **slow** for **jupiter-2**
  + Set it to **fast** for **jupiter-3**
* (T205 / 1 pts) Explore the manifest **/files/jupiter/t205-in.yaml**
  + Modify it in such a way that if deployed, the workload to go on the node with label **exam** set to **fast** and save the new manifest as **/files/jupiter/t205-out.yaml**
  + Deploy it to the cluster
* (T206 / 1 pts) Explore the manifest **/files/jupiter/t206-in.yaml**
  + Modify it in such a way that if deployed, the workload to go on the node named **jupiter-2** and save the new manifest as **/files/jupiter/t206-out.yaml**
  + Deploy it to the cluster
* (T207 / 4 pts) Explore the **banana** namespace. There should be a pair of a pod and service. They are created out of the **/files/jupiter/t207-in.yaml** manifest. The problem is that the pod (**banana-pod-1**) is not in running state and the service (**banana-svc**) does not have any endpoints. Your mission is to:
  + Correct these issues and save the changes as **/files/jupiter/t207-out.yaml** manifest
  + Make sure that the deployed objects are in a good shape (they reflect the corrections)
  + Add a second pod, in the new manifest file, of the same type but change the image tag to green and name it **banana-pod-2**
  + Make sure that the new pod is present in the service endpoints list
* (T208 / 4 pts) Explore the manifest **/files/jupiter/t208-in.yaml** and
  + Extend it to also include
    - A definition of a namespace named **cherry**
    - and a definition of a service named **cherry-svc** of type **NodePort** and port set to **32208**
    - and save it as **/files/jupiter/t208-out.yaml**
  + Deploy the manifest

#### Venus Cluster [17 pts]

* (T301 / 5 pts) Install the missing system components (**kubeadm**, **kubelet**, and **kubectl**) on the **venus-2** node and make sure that their version is aligned with the version installed on the **venus-1** node
* (T302 / 3 pts) Join the **venus-2** node to the **Venus Cluster**
* (T303 / 2 pts) Deploy **Antrea** pod network plugin on the **Venus** **Cluster**
* (T304 / 3 pts) Modify the configuration of the **Venus** **Cluster** in such a way to allow workload to be placed on the **control plane** node
* (T305 / 4 pts) Explore the manifest **/files/venus/t305-in.yaml** and
  + Change it in such a way (save it under **/files/venus/t305-out.yaml**) that the described pod is **deployed on every node of the cluster**
  + Deploy the resulting manifest

## Sample Solution

Okay, for starters we must read the tasks and prepare our strategy how we will tackle them

Let’s imagine, that we read it twice and came up with this:

### Venus Cluster

Start from the **exam station** and open a SSH session to the **venus-1** machine

#### T301

Now, check the nodes of the cluster

**kubectl get nodes**

Okay, the version is **1.21.6** and there is just one node – the control plane node

Close the session and open a new session to the **venus-2** machine

Check if there is the required repository

**grep kubernetes /etc/apt/sources.list**

Nothing. Now, check here

**ls -al /etc/apt/sources.list.d/**

Aha, here it is. Refresh the repository and package information

**sudo apt-get update**

And install the three system packages with

**sudo apt-get install kubelet=1.21.6-00 kubeadm=1.21.6-00 kubectl=1.21.6-00**

Don’t forget to mark them for hold

**sudo apt-mark hold kubelet kubeadm kubectl**

#### T302

Now, we must find a way to join the node to the cluster

But how, if do not know the join token? Perhaps, create a new one

Return on **venus-1** and check if there are any active

**kubeadm token list**

Even if there is, it is easier to create a new join token

**kubeadm token create --print-join-command**

Return on the **venus-2** and join it to the cluster

**sudo kubeadm join 192.168.81.231:6443 --token <token> --discovery-token-ca-cert-hash <hash>**

Now, return to the **venus-1** node

Check the status

**kubectl get nodes**

#### T303

We must do one more thing – install the pod network plugin – **Antrea** in our case

Let’s do it. We must execute

**kubectl apply -f https://raw.githubusercontent.com/antrea-io/antrea/main/build/yamls/antrea.yml**

After a while, the resources will be deployed, and all nodes will be in ready state

Return on the **exam station**

Check the available contexts

**kubectl config get-contexts**

Switch to the **Venus Cluster**

**kubectl config use-context venus-admin@venus**

Check the nodes with

**kubectl get nodes -o wide**

#### T304

Next, check the taints for all nodes

**kubectl describe node | grep Taints**

Now, remove the taint of the control plane node with

**kubectl taint nodes venus-1 node-role.kubernetes.io/master:NoSchedule-**

Check again the taints for all nodes

**kubectl describe node | grep Taints**

We are done. Let’s continue

#### T305

Check the **/files/venus/t305-in.yaml** manifest

It is a simple one. We must enrich it with a **DeamonSet**

First, copy it to **/files/venus/t305-out.yaml**

Now open it for editing and make sure that is looks like

apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: ds305

spec:

  selector:

    matchLabels:

      app: ds305

  template:

    metadata:

      labels:

        app: ds305

    spec:

      containers:

      - image: alpine

        name: main

        args:

        - /bin/sh

        - -c

        - sleep 86400

Save it and close it

Deploy it to the cluster

**kubectl apply -f t305-out.yaml**

Check if there are pods and they are on every node

**kubectl get pods -o wide**

Done with this task and the cluster as a whole

### Mars Cluster

Okay, check again the available contexts

**kubectl config get-contexts**

Switch to the **Mars Cluster**

**kubectl config use-context mars-admin@mars**

Check the nodes with

**kubectl get nodes -o wide**

#### T101

Check the contents of the **animals** namespace

**kubectl get pods,svc -n animals**

Okay, there are two pods and two services – one for each pod

Now, check what ingress class exists

**kubectl get ingressclass**

Okay, it is **HAProxy**

Let’s create the manifest. Create a file **/files/mars/t101-out.yaml** with the following content

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: pets-ingress

  namespace: animals

  annotations:

    haproxy.org/path-rewrite: "/"

spec:

  ingressClassName: haproxy

  rules:

  - host: pets.lab

    http:

      paths:

      - path: /cat

        pathType: Prefix

        backend:

          service:

            name: cat-svc

            port:

              number: 80

      - path: /dog

        pathType: Prefix

        backend:

          service:

            name: dog-svc

            port:

              number: 80

Save it and close it and then deploy it with

**kubectl apply - f /files/mars/t101-out.yaml**

Check that it has been deployed correctly

**kubectl get ingress -n animals**

**kubectl describe ingress pets-ingress -n animals**

Check the ingress **NodePort** port number

**kubectl get service -n haproxy-controller**

Then check the ingress is working with

**curl http://pets.lab:32221/cat**

**curl http://pets.lab:32221/dog**

Everything is working

#### T102

Let’s check if there is a pod and what is its state

**kubectl get pods -n tiger**

Indeed, there is one and it is not in a good shape

Let’s copy the manifest as **/files/mars/t102-out.yaml** and explore it

Aha, the image name is wrong, let’s set it to **alpine** instead of **alpain** and redeploy the pod

**kubectl delete -f /files/mars/t102-out.yaml**

**kubectl apply -f /files/mars/t102-out.yaml**

And check again

**kubectl get pods -n tiger**

Still not working. Let’s check again the manifest

Hm, the tag seems not correct (especially if we do a cross check with **Docker Hub**). Let’s remove it (effectively we will target the **latest**) and check again

**kubectl apply -f /files/mars/t102-out.yaml**

Now, it is working at least it seams so. Let’s wait a while and watch with

**kubectl get pods -n tiger -w**

We will see that it is restarting, and this is not okay

Let’s check once more the manifest. Aha, it seems that the liveness probe is causing this. Let’s correct it by applying an initial delay of at least 65 seconds (as we can see that the container is waiting for 60 seconds and then creates the file which the liveness probe is looking for). Redeploy it

**kubectl delete -f /files/mars/t102-out.yaml**

**kubectl apply -f /files/mars/t102-out.yaml**

Now, even if we wait and watch a bit, we won’t notice any restarts

#### T103

Okay, let’s prepare the folder structure first

**mkdir -p /files/mars/t103/{base,overlays}**

**mkdir -p /files/mars/t103/overlays/{blue,green}**

Now, copy the source file under new name and to a new place

**cp /files/mars/t103-in.yaml /files/mars/t103/base/**

Navigate to the **/files/mars/t103/** folder

Then create a **base/kustomization.yaml** file for the **base** variant with the following content

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

resources:

- t103-in.yaml

Test that the **base** variant is working

**kustomize build base/**

It should display a valid manifest. You can even test is if you want

Let’s continue with the creation of the **blue** variant

Create an **overlays/blue/kustomizatoin.yaml** file with the following initial content

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

namePrefix: blue-

commonLabels:

  variant: blue

resources:

- ../../base

patchesStrategicMerge:

- cust-dpl.yaml

- cust-svc.yaml

Now, create the **overlays/blue/cust-dpl.yaml** file to adjust the replica count with the following content

apiVersion: apps/v1

kind: Deployment

metadata:

  name: kust

spec:

  replicas: 3

And the **overlays/blue/cust-svc.yaml** file to adjust the node port with the following content

apiVersion: v1

kind: Service

metadata:

  name: kust

spec:

  ports:

  - nodePort: 31103

    port: 80

    protocol: TCP

Now, only the image tag is still the same. Let’s change it by entering the **blue** variant folder and execute

**kustomize edit set image shekeriev/k8s-environ:latest=shekeriev/k8s-environ:blue**

Now, return to the **/files/mars/t103** folder and test the **blue** variant

**kustomize build overlays/blue/**

It seems just fine. Now, prepare the **green** variant by duplicating the **kustomization.yaml** file

**cp overlays/blue/kustomization.yaml overlays/green/**

Adjust it to match the following

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

namePrefix: green-

commonLabels:

  variant: green

resources:

- ../../base

patchesStrategicMerge:

- cust-svc.yaml

images:

- name: shekeriev/k8s-environ:latest

  newName: shekeriev/k8s-environ

  newTag: green

Then create the **overlays/green/cust-svc.yaml** file by copying the one from the **blue** variant

**cp overlays/blue/cust-svc.yaml overlays/green/**

And to adjust the node port with the following content

apiVersion: v1

kind: Service

metadata:

  name: kust

spec:

  ports:

  - nodePort: 32103

    port: 80

    protocol: TCP

Now, make sure you are in the **/files/mars/t103** folder and test the **green** variant

**kustomize build overlays/green/**

It seems fine. So, we are almost ready

Now, deploy the two variants using the two approaches (not that we have to, but because we can 😉):

**kustomize build overlays/blue/ | kubectl apply -f -**

**kubectl apply -k overlays/green/**

Check that all is up and running

**kubectl get pods,svc**

Yes, all is working

#### T104

Let’s first check the situation

**kubectl get pods,deployment,rs -n cherry**

Aha. Perhaps, there is a problem with the image. It is not the only one, as we can see that there are 5 desired replicas and we have only 3

Let’s first deal with the image issue and then we will handle the replica issue (which is the main purpose of the task)

As we do not have starting manifest, we can either generate one, or edit directly the configuration of the deployment. Let’s go with the first option

**kubectl get deployment -n cherry cherry -o yaml > t104.yaml**

First, use it to delete the resources

**kubectl delete -f t104.yaml**

Then open it for editing and make sure you clean it to the following

apiVersion: apps/v1

kind: Deployment

metadata:

  name: cherry

  namespace: cherry

spec:

  replicas: 5

  selector:

    matchLabels:

      app: cherry

  template:

    metadata:

      labels:

        app: cherry

    spec:

      containers:

      - args:

        - /bin/sh

        - -c

        - sleep 86400

        image: alpain:3.22

        name: main

Aha, the same “wrong” image – **alpain:3.22**. Change it to **alpine** and save the manifest. Then try do deploy it

**kubectl apply -f t104.yaml**

Check again the resources

**kubectl get pods,deployment,rs -n cherry**

Yes, it is working. One issue down, unfortunately not the one the task wants

We can see that there is still issue with the replica count

Usually, this is cause by some kind of restriction. Perhaps, a quota

Let’s check if there are any

**kubectl get quota -n cherry**

Yes, we are good. There is the reason. Save it to file **/files/mars/t104-reason.txt** with the following content

**ResourceQuota:cherry**

And then correct the deployment to fit within the quota – change the replica count to 3

**kubectl edit deployment -n cherry**

After a few seconds if we check

**kubectl get pods,deployment,rs -n cherry**

We will see that all is nice and clean. We are done here

#### T105

This one is easy. First, create the target manifest file

**cp /files/mars/t105-in.yaml /files/mars/t105-out.yaml**

Then open it for editing and adjust it to match this

apiVersion: batch/v1

kind: CronJob

metadata:

  name: five-job

spec:

  schedule: "\*/5 \* \* \* \*"

  jobTemplate:

    spec:

      template:

        spec:

          restartPolicy: OnFailure

          containers:

          - image: alpine

            name: main

            args:

            - /bin/sh

            - -c

            - sleep 60

Save and close it. Deploy it to the cluster

**kubectl apply -f /files/mars/t105-out.yaml**

And check if the resources are created

**kubectl get pods,cronjobs**

After a while a new pod will be created. We do not have to wait, instead we will continue

#### T106

Check if there is something deployed in the **fortress** namespace

**kubectl get pods,sa,role,rolebinding -n fortress**

Okay, nothing yet (just the **default** service account)

Let’s start creating the required objects

Create a service account

**kubectl create serviceaccount observer -n fortress**

Next, create the role

**kubectl create role looknotouch -n fortress --verb=get --resource=pods**

Then, we can create the role binding

**kubectl create rolebinding looknotouch -n fortress --role=looknotouch --serviceaccount=fortress:observer**

The last thing is to change the manifest. Let’s do a copy first

**cp /files/mars/t106-in.yaml /files/mars/t106-out.yaml**

And then, make sure that we add the **serviceAccount** clause so the manifest to look like

apiVersion: v1

kind: Pod

metadata:

  name: observer

  namespace: fortress

spec:

  serviceAccount: observer

  containers:

  - image: alpine

    name: main

    args:

    - /bin/sh

    - -c

    - sleep 86400

Save and close the file and then deploy it to the cluster

**kubectl apply -f /files/mars/t106-out.yaml**

And check if the pod appears in the set of resources

**kubectl get pods,sa,role,rolebinding -n fortress**

Yes, it is there. Let’s see if the service account has been set correctly

**kubectl get -n fortress pod/observer -o yaml | grep serviceAccount**

Yes. All is fine. We are done here

### Jupiter Cluster

Make sure that you are working on the **exam** station

Let’s check again the available contexts

**kubectl config get-contexts**

And switch to the **Jupiter Cluster**

**kubectl config use-context jupiter-admin@jupiter**

Check that we have connectivity with the cluster by asking for the nodes

**kubectl get nodes -o wide**

All is set and we can start working on the tasks

For the sake of the experiment, let’s start not with the first, but with the second task

#### T202

Open a browser tab and navigate to <https://artifacthub.io>

Search for **apache**

Usually, one of the first results is the **Apache** chart provided by **Bitnami**. Click on it

Check the instructions and add the repository

**helm repo add bitnami https://charts.bitnami.com/bitnami**

Explore the list of available namespaces

**kubectl get ns**

No, the **kiwi** one is not there. So, let’s create it

**kubectl create ns kiwi**

Return to the page for the chart and explore the information there

It appears that we must set custom values for the **service.type**, **service.nodePorts.http** and the **htdocsConfigMap** parameters

First, we must prepare the custom configuration map

**kubectl create configmap -n kiwi exam-httpd-cm --from-literal=index.html="Kubernetes+Helm=Fun"**

Then, our install command will become

**helm install exam-httpd -n kiwi bitnami/apache --set service.type=NodePort --set service.nodePorts.http=32202 \**

**--set htdocsConfigMap=exam-httpd-cm**

Check that all resources are there

**kubectl get pods,svc,cm -n kiwi**

And check that the custom web page has been deployed

**curl http://jupiter-1:32202**

Everything is okay

#### T203

Check what we have in the cucumber namespace

**kubectl get pods,secret -n cucumber**

Okay, let’s explore the pod and see how the secret is being consumed by it

**kubectl get pod/cucumber -n cucumber -o yaml**

It appears that it is consumed as an environment variable. So, we can execute

**kubectl exec -it pod/cucumber -n cucumber -- sh -c set**

And see the secret in plain text

Alternatively, we can execute the following command to explore the secret itself

**kubectl get secret -n cucumber cucumber-secret -o jsonpath={.data.scretfact} | base64 --decode**

We should see the same. In this case - **U r h@k3r**

Don’t forget to store the value in the required file. In fact, using a modified version of the second command, we can automate the process

**kubectl get secret -n cucumber cucumber-secret -o jsonpath={.data.scretfact} | base64 --decode > /files/jupiter/t203-secret.txt**

One more thing. We must change the secret

Let’s first generate the encoded value of the new one with

**echo 'Cucumbers are green' | base64**

Then open the secret for editing with

**kubectl edit secret -n cucumber cucumber-secret**

And change the value of the **scretfact** with the encoded value - **Q3VjdW1iZXJzIGFyZSBncmVlbgo=**

Save and close the file

Just to be sure, check the current state of the secret

**kubectl get secret -n cucumber cucumber-secret -o jsonpath={.data.scretfact} | base64 --decode**

All is fine. We are done here

#### T204

Use the following commands to set the required labels

**kubectl label node jupiter-2 exam=slow**

**kubectl label node jupiter-3 exam=fast**

And check that the labels are set

**kubectl get nodes --show-labels**

Sure, they are

#### T205

First, create a copy of the starting file

**cp /files/jupiter/t205-in.yaml /files/jupiter/t205-out.yaml**

Now, open it and make sure it looks like

apiVersion: v1

kind: Pod

metadata:

  name: pod205

spec:

  nodeSelector:

    exam: fast

  containers:

  - image: alpine

    name: main

    args:

    - /bin/sh

    - -c

    - sleep 86400

Save it and close it and then deploy it

**kubectl apply -f /files/jupiter/t205-out.yaml**

And then check, that the pod indeed went to the required node

**kubectl get pods -o wide**

Yes, it went on the **jupiter-3** node which matches the requirement – **exam=fast**

#### T206

First, create a copy of the starting file

**cp /files/jupiter/t206-in.yaml /files/jupiter/t206-out.yaml**

Now, open it and make sure it looks like

apiVersion: v1

kind: Pod

metadata:

  name: pod206

spec:

  nodeName: jupiter-2

  containers:

  - image: alpine

    name: main

    args:

    - /bin/sh

    - -c

    - sleep 86400

Save it and close it and then deploy it

**kubectl apply -f /files/jupiter/t206-out.yaml**

And then check, that the pod indeed went to the required node

**kubectl get pods -o wide**

Yes, it went on the **jupiter-2** node which was exclusively stated in the manifest

#### T207

First, check what we have in the **banana** namespace

**kubectl get pods,svc -n banana**

Most probably there is an issue with the image

Make a copy of the provided manifest

**cp /files/jupiter/t207-in.yaml /files/jupiter/t207-out.yaml**

Now, open it and explore it

Aha, there is an issue with the image indeed. It is set to **shekeriev/k8s-emviron:blue** and it must be **shekeriev/k8s-environ:blue**

Delete the old pod and deploy the one with the new image

**kubectl delete pod banana-pod-1 -n banana**

**kubectl apply -f /files/jupiter/t207-out.yaml**

Now, check the resources again

**kubectl get pods,svc -n banana**

Okay, the pod is in running state. Let’s see if it is part of the endpoints list of the service

**kubectl describe service banana-svc -n banana**

No, it is not there. Let’s explore the manifest (**/files/jupiter/t207-out.yaml**)again

Aha, it seems that the label of the pod is not set. Make sure that the current version looks like

apiVersion: v1

kind: Pod

metadata:

  name: banana-pod-1

  namespace: banana

  labels:

    app: banana

spec:

  containers:

  - image: shekeriev/k8s-environ:blue

    name: main

    env:

    - name: FRUIT

      value: "Did you know that bananas are yellow and curved? :)"

    - name: FOCUSON

      value: "FRUIT"

---

apiVersion: v1

kind: Service

metadata:

  name: banana-svc

  namespace: banana

spec:

  ports:

  - port: 80

    protocol: TCP

  selector:

    app: banana

Save and close the file and redeploy it

**kubectl apply -f /files/jupiter/t207-out.yaml**

Check again the endpoints of the service

**kubectl describe service banana-svc -n banana**

Finally, the pod is there. There is one more thing – we must add another pod with the same image but different tag

Add it as a separate resource in the same manifest

The final version of the **/files/jupiter/t207-out.yaml** should be

apiVersion: v1

kind: Pod

metadata:

  name: banana-pod-1

  namespace: banana

  labels:

    app: banana

spec:

  containers:

  - image: shekeriev/k8s-environ:blue

    name: main

    env:

    - name: FRUIT

      value: "Did you know that bananas are yellow and curved? :)"

    - name: FOCUSON

      value: "FRUIT"

---

apiVersion: v1

kind: Pod

metadata:

  name: banana-pod-2

  namespace: banana

  labels:

    app: banana

spec:

  containers:

  - image: shekeriev/k8s-environ:green

    name: main

    env:

    - name: FRUIT

      value: "Did you know that bananas are yellow and curved? :)"

    - name: FOCUSON

      value: "FRUIT"

---

apiVersion: v1

kind: Service

metadata:

  name: banana-svc

  namespace: banana

spec:

  ports:

  - port: 80

    protocol: TCP

  selector:

    app: banana

Save and close the file and redeploy the resources

**kubectl apply -f /files/jupiter/t207-out.yaml**

Check that all is there

**kubectl get pods,svc -n banana**

And then that both pods are part of the endpoints list of the service

**kubectl describe service banana-svc -n banana**

All is set. Done

#### T208

This one is easy; we have to extend the manifest with a namespace and a service

First, let’s make a copy of the manifest

cp /files/jupiter/t208-in.yaml /files/jupiter/t208-out.yaml

Then, make sure that it includes the required objects and looks like

apiVersion: v1

kind: Namespace

metadata:

  name: cherry

---

apiVersion: v1

kind: Pod

metadata:

  name: t208

  namespace: cherry

  labels:

    app: cherry

spec:

  containers:

  - image: shekeriev/k8s-environ

    name: main

    env:

    - name: FRUIT

      value: "Did you know that cherries are red and tasty? :)"

    - name: FOCUSON

      value: "FRUIT"

---

apiVersion: v1

kind: Service

metadata:

  name: cherry-svc

  namespace: cherry

spec:

  type: NodePort

  ports:

  - port: 80

    protocol: TCP

    nodePort: 32208

  selector:

    app: cherry

Don’t forget the label. Even though not explicitly stated in the task, it is a required attribute

Save and close the file and deploy it to the cluster

**kubectl apply -f /files/jupiter/t208-out.yaml**

Then check that all resources are in place

**kubectl get pods,svc -n cherry**

Yes, they are. We can even check that the application is working

**curl http://jupiter-1:32208**

Yes, it is. We are getting better and better 😊

One last task – the first one in this section. Let’s do it

#### T201

First, let’s explore the objects in these namespaces

**kubectl get pods,svc -n apple**

**kubectl get pods,svc -n orange**

**kubectl get pods,svc -n apricot**

Everything seems fine

Let’s try the three applications by trying to reach their services

**curl http://jupiter-1:30901**

**curl http://jupiter-1:30902**

**curl http://jupiter-1:30903**

All three are working and reporting that they can reach the others as well

In order to fulfill the requirements of the task, we must create an **ingress network policy** in the **apple** namespace

First, let’s see if the orange namespace has any labels

**kubectl get ns orange --show-labels**

No, there is just the standard one

**kubernetes.io/metadata.name=orange**

We will use it

Create a manifest **/files/jupiter/t201-out.yaml** with the following content

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

  name: access-apple

  namespace: apple

spec:

  podSelector: {}

  policyTypes:

  - Ingress

  ingress:

  - from:

    - namespaceSelector:

        matchLabels:

          kubernetes.io/metadata.name: orange

Save it and close it. Then deploy it to the cluster

**kubectl apply -f /files/jupiter/t201-out.yaml**

Now, check that the resource is there

**kubectl get netpol -n apple**

And see how it is translated by the cluster

**kubectl describe netpol access-apple -n apple**

It seems that it matches what we wanted to do

Let’s test it

**curl --connect-timeout 5 http://jupiter-1:30901**

**curl --connect-timeout 5 http://jupiter-1:30902**

**curl --connect-timeout 5 http://jupiter-1:30903**

As we can see, the first service is not accessible. The second reports that can reach all others including the one from the **apple** namespace. And the third reports that it cannot reach the service in the **apple** namespace

So, we managed to do this one as well

We are done with everything. We should be proud of ourselves 😊