# Practice M8: Exam Prep (Option #3)

During this practice we will assume that we are working in **Windows** environment. It could be either a physical machine or a virtual one

Additionally, you must have **Vagrant** and **VirtualBox** (preferably) or another virtualization solution installed

All steps can be executed in most **Linux** distributions and/or **macOS** environment as well

The infrastructure can be built on-premise or in the cloud (AWS, GCP, Azure). Any combination of tools is acceptable

## Assignment

Prepare a clustered environment to host a containerized web application. Application’s images must be built in an automated way – check every 3 minutes and if there are changes, build the images, publish them, and then re-deploy them. The cluster should be monitored – a dashboard, showing the utilization must be created

## Possible Solution

We will implement one possible solution. It will include the following set of technologies:

* it will be hosted locally in **VirtualBox**
* the infrastructure and cluster will be built with **Minikube**
* the cluster will be based on **Kubernetes** and will have one node
* for monitoring we will use **Elastic Stack**
* logs and metrics will be collected with **Filebeat** and **Metricbeat**
* **Jenkins** and **Docker** will be used for image building

### Infrastructure & Cluster

For this part, we will use **Minikube**. But instead using it will its default configuration parameters, we will adjust them. Our focus will be on parameters like the number of vCPUs and memory size.

Let’s initiate the building process:

* Go to folder **M8-3** if not there already and execute:

(For Windows)

**minikube start --driver=virtualbox --cpus=4 --memory=8192**

(For Linux)

**minikube start --cpus=4 --memory=8192**

* Of course, you can change the values according to the availability of resources on the hardware you possess
* Wait until everything is up and running. This can take up to 5 minutes
* **Minikube** will prepare the environment, so we start using our single node all-in-one **Kubernetes** cluster immediately
* Now we can check if we can communicate with the cluster:

**kubectl cluster-info**

* And then ask for additional information:

**kubectl get nodes**

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

### Monitoring

Now that we have a working infrastructure, it is time to take care of the monitoring part. For this task we will deploy an **Elastic Stack**. We will reuse the images created and published during the first part of this practice.

First, we could adjust the **Elastic** deployment files if needed:

* Go to folder **M8-3/elastic-stack**
* Examine and adjust if needed both **\*deployment.yml** and **\*service.yml** files for every component
* Adjust the **logstash-config.yml** file if needed
* Go one folder up and execute:

**kubectl create -f elastic-stack/ -R --namespace=default**

* Monitor the pods creation with

**kubectl get pods**

* Once, all pods are in running state, copy the name of the **Kibana** pod and follow its logs to know when it is ready (it can take up to 15 minutes):

**kubectl logs -f kibana-xxxxxxxx-yyyyy**

It is time to deploy the two **Beat** pods. First, we will deploy the **Filebeat**:

* Go to folder **M8-3/filebeat**
* Examine and adjust the files if needed
* Being in the **M8-3** folder, execute:

**kubectl create -f filebeat/ -R --namespace=default**

And then we can deploy the **Metricbeat** pods:

* Go to folder **M8-3/metricbeat**
* Examine and adjust the files if needed
* Being in the **M8-3** folder, execute:

**kubectl create -f metricbeat/ -R --namespace=default**

Once the **Kibana** is up and running, we should examine where we can reach it:

**minikube service list**

Now, we can go to **Kibana** (http://<minikube-ip>:30000/), create the **Index Pattern**, then explore the data, and finally, create one or more visualizations, and put them in a dashboard.

IP address of the Minikube machine could be retrieved with **minikube ip**

In this part there is plenty room for improvement. For example, we can add/enable additional modules of **Metricbeat**, and/or change the **grok filter** in the **Logstash** configuration.

Valid option for monitoring and control is to utilize the **Kubernetes Dashboard**. It should be deployed by default:

* Let’s check if the module (add-on) is enabled:

**minikube addons list**

* If the **Dashboard** add-on is not enabled, we can do it:

**minikube addons enable dashboard**

* We can enable also **Metricsserver** if we like
* In order to open the **Dashboard**, we must execute:

**minikube dashboard**

* It will open a browser tab pointing to the **Dashboard**
* Explore a bit. For example, check different namespaces, pods, services, deployments, and etc.

### CD/CI Preparation

We will deploy **Jenkins** in our **Kubernetes** **Minikube** cluster. And instead of doing it manually, we will use **Helm** package manager:

* Create a separate namespace:

**kubectl create namespace jenkins**

* Add the **Jenkins** repository

**helm repo add jenkins https://charts.jenkins.io**

* Refresh **Helm** repository information if needed with:

**helm repo update**

* Install **Jenkins**:
  + With its defaults, which is not suitable for our setup (do not execute this one):

**helm install jenkins --namespace jenkins jenkins/jenkins**

* + For our setup we will change just the way we communicate with it – we will use **NodePort**:

**helm install jenkins --namespace jenkins --set controller.serviceType=NodePort --set controller.nodePort=30001 jenkins/jenkins**

* We will go with the last option
* Get information for all pods in order to be sure that the Jenkins is operational:

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

* Or we can ask for the status of **Jenkins** this way:

**helm status jenkins --namespace jenkins**

* Now, we should add one binding:

**kubectl create clusterrolebinding jenkins --clusterrole=cluster-admin --serviceaccount=jenkins:default**

* And one configuration map that will point to our **Docker Registry** which in our case is **Docker Hub**:

(For Windows)

**kubectl create configmap docker-config --from-file=%HOMEPATH%\.docker\config.json -n jenkins**

(For Linux)

**kubectl create configmap docker-config --from-file=$HOME/.docker/config.json -n jenkins**

* Get the password for the admin user:

(For Windows)

**kubectl get secret --namespace jenkins jenkins -o jsonpath="{.data.jenkins-admin-password}"**

Then use a tool (for example <https://www.base64decode.org/>) to decode the string

(For Linux)

**printf $(kubectl get secret --namespace jenkins jenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 --decode);echo**

* Alternative way for obtaining the credentials will be to use the **Kubernetes Dashboard** and navigate to the **Jenkins** pod
* Get information about the service:

**kubectl get svc --namespace jenkins jenkins**

* Alternative option would be to execute:

**minikube service list**

* Open a browser tab and navigate to http://<minikube-ip>:30001
* Once in, we must configure **Jenkins**. Go to **Manage Jenkins** and then **Manage Nodes and Clouds**
  + Click **Configure Clouds**
  + It would be enough to ensure that both **Kubernetes Namespace** and **Kubernetes Pod Template Namespace** are set to the namespace in which we deployed **Jenkins**. In our case this should be **jenkins**
* Click on **Save**

#### CD/CI – build and apply – with Git, Jenkins and Docker

We will create the same job, as we did in the last section of **M8 Exam Prep Practice**:

* Go to **New Item**
* Select **Pipeline** for type
* In the **Enter an item name** set some name, for example **Pipeline-Docker-Apply**
* In the pipeline script enter (or copy it from the **M8-3/jenkins/job-pipeline-docker-apply.txt** file):

**def label = "docker-${UUID.randomUUID().toString()}"**

**podTemplate(label: label, yaml: """**

**apiVersion: v1**

**kind: Pod**

**spec:**

**containers:**

**- name: docker**

**image: docker:20.10.8**

**command: ['cat']**

**tty: true**

**volumeMounts:**

**- name: dockersock**

**mountPath: /var/run/docker.sock**

**- name: docker-config**

**mountPath: /root/.docker**

**- name: kubectl**

**image: lachlanevenson/k8s-kubectl:v1.21.4**

**command: [cat]**

**tty: true**

**volumeMounts:**

**- name: dockersock**

**mountPath: /var/run/docker.sock**

**- name: docker-config**

**mountPath: /root/.docker**

**volumes:**

**- name: dockersock**

**hostPath:**

**path: /var/run/docker.sock**

**- name: docker-config**

**configMap:**

**name: docker-config**

**"""**

**) {**

**def imagetag = new Date().format('yyyyMMdd.HHmmss')**

**def image = "shekeriev/dofepapp:${imagetag}"**

**node(label)**

**{**

**stage('Build Docker image')**

**{**

**git branch: 'main', url: 'https://github.com/shekeriev/dofepapp.git'**

**container('docker')**

**{**

**sh "docker build -t ${image} app/."**

**}**

**}**

**stage ("Push")**

**{**

**container('docker')**

**{**

**sh "docker push ${image}"**

**}**

**}**

**stage ("Apply the changes with kubectl")**

**{**

**container('kubectl')**

**{**

**sh "sed 's/%IMAGE-PLACEHOLDER%/${imagetag}/g' -i yaml/application.yaml"**

**sh "kubectl apply -f yaml/application.yaml"**

**}**

**}**

**}**

**}**

* Click **Save**
* Click on **Build Now**
* Check the result either on the command line and/or in the browser

### Final touches

We can install the Git Hub plugin and set up the project URL and the schedule

### Clean up

Don’t forget to clean up. With this approach, there is just one step required 😊

**minikube delete**