# Practice M8: Exam Prep – Option 2

During this practice we will assume that we are working in Linux environment. It could be a physical machine or a virtual one. The distribution of choice is not that important, but it will be better to stick to some of the well supported distributions.

Most of the steps can be executed in Windows and/or macOS environment as well, either directly or in a VM.

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 Vagrant + Kubeadm + Shell Scripts
* the cluster will be based on Kubernetes and will have one master and two worker nodes
* 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 the **Vagrantfile** created during **Part 3** of **Module 2**. Basically, we can use it as it is, but we will do two changes:

* As the things are developing, we must change the URL for the **Kubernetes Dashboard** installed while provisioning the first node. Then new URL should be (on line 97):

**kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml**

* Then we will change the solution used for **POD network**:
  + instead of using **Flannel** (<https://github.com/coreos/flannel>), this time we will use:
  + **Wave Net** (<https://github.com/weaveworks/weave/blob/master/site/kubernetes/kube-addon.md>)
* Then new command on line 94 will become:

**kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"**

Of course, we could adjust other settings as well – for example, the number of vCPUs, memory size, or the number of nodes. Let’s save it and continue with the next steps.

Once, we are okay with the configuration, we must initiate the building process:

* Go to folder **M8/M8-2** if not there already and execute:

**vagrant up**

* Wait until everything is up and running. This can take 10-15 minutes
* Get the configuration from the master node in order to be able to use **kubectl** locally:

**scp vagrant@192.168.99.101:.kube/config .**

* Now, either **replace your existing** one, or just **store it as an additional configuration** and then set the environment variable … to point to it when needed. For the purpose of this exercise, we will overwrite our existing configuration:

**mv config ~/.kube/config**

* 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/M8-2/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**

* Check 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/M8-2/filebeat**
* Examine and adjust the files if needed
* Being in the **M8/M8-2** folder, execute:

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

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

* Go to folder **M8/M8-2/metricbeat**
* Examine and adjust the files if needed
* Being in the **M8/M8-2** folder, execute:

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

Now, we can go to **Kibana** (<http://192.168.99.101:30000/>), create the **Index Pattern**, then explore the data, and finally, create one or more visualizations, and put them in a dashboard.

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 the **Vagrantfile**:

* One viable option to access it is to execute:

**kubectl proxy**

* Now, open a second terminal and execute:

**kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')**

* Or go to folder **M8/M8-2/vagrant** and examine the contents of the file **amin-user-token.txt**
* Now, that you know the token, open a browser tab and navigate to:

**http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login**

* Explore a bit. For example, check different namespaces, pods, services, deployments, and etc.

#### CD/CI Preparation

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

* Create a separate namespace:

**kubectl create namespace jenkins**

* Create a service account and binding:

**kubectl create serviceaccount --namespace kube-system tiller**

**kubectl create clusterrolebinding tiller-cluster-admin --clusterrole=cluster-admin --serviceaccount=kube-system:tiller**

* Initialize **Helm**:

**helm init --service-account tiller**

* Install **Jenkins**:
  + With its defaults, which is not suitable for our setup:

**helm install --name jenkins --namespace jenkins stable/jenkins**

* + With just persistence turned off:

**helm install --name jenkins --namespace jenkins --set Persistence.Enabled=false stable/jenkins**

* + With **NodePort** set to on, instead of the default **LoadBalancer**, and Persistence set to off:

**helm install --name jenkins --namespace jenkins --set Master.ServiceType=NodePort --set Persistence.Enabled=false stable/jenkins**

* + With **NodePort** set to a custom port and persistence set to off:

**helm install --name jenkins --namespace jenkins --set Master.ServiceType=NodePort --set Master.NodePort=30001 --set Persistence.Enabled=false stable/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**

* Now, we should add one more 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**:

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

* Get the password for the admin user:

**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**

* Open a browser tab and navigate to <http://192.168.99.101:30001>
* Once in, resolve the issue with the plugins by updating them and restarting the **Jenkins** instance
* Next, we must configure **Jenkins**. Go to **Manage Jenkins** and then **Configure System**
  + 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/M8-2/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:1.11**

**command: ['cat']**

**tty: true**

**volumeMounts:**

**- name: dockersock**

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

**- name: docker-config**

**mountPath: /root/.docker**

**- name: kubectl**

**image: lachlanevenson/k8s-kubectl:v1.8.0**

**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/k8s-jenkins:${imagetag}"**

**node(label)**

**{**

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

**{**

**git 'https://github.com/shekeriev/simple-docker-image.git'**

**container('docker')**

**{**

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

**}**

**}**

**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.yml"**

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

**}**

**}**

**}**

**}**

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

#### Clean up

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

**vagrant destroy --force**