Lab 05: Kubernetes

The exercises marked with * are obligatory to complete before the upcoming week's lecture.

The purpose of this lab is to provide the student with an insight into the application of Kubernetes and its configuration. The lab is structured in a way, where the student is assisted through.

Remember to write useful commands in your cheat sheets.

Table of content

Exercise 01* **1**

Exercise 02* **3**

~~**Exercise 03***~~ ~~**4**~~

Exercise 04* **5**

Exercise 05* **7**

PodsS 7

Deployments 7

Services 7

NodePort 7

Spekt8 8

Exercise 06* **8**

Kubernetes Manifest 8

Exercise 07 **10**

Exercise 01*

In this exercise the student is going to self-sign a SSL certificate. The theory behind this exercise was presented last week. (in lecture 4). This exercise only focuses on the creation of a signed certificate and not how to configure a TLS server to use it.

Create your own root certificate authority keypair, that will be used to sign certificate requests.
The private key is *private* meaning it should only be available to you.

```
$ openssl genrsa -des3 -out rootCA.key 2048
```

Generate the certificate containing the public key from the private key.(rootCa.key)
The public key is *public* meaning it can be shared with everybody.

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

After the CA has been generated, each server that needs a certificate from you CA will have to create a private key and a certificate signing request (CSR)

```
$ openssl req -new -sha256 -nodes -out server.csr -newkey rsa:2048 -keyout <DOMAIN>.key
```

(You don't have to provide challenge password)

A file, v3.ext, has to be created holding configuration information about the certificate and the domain name of the server.

v3.ext:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
```

Issue the certificate using the certificate signing request and the CA's certificate and private key.

```
$ openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out <DOMAIN>.crt -days 1000 -sha256 -extfile v3.ext
```

To see what the x.509 (The name of the certificate standard) certificate contains:

```
$ openssl x509 -in <DOMAIN>.crt -text -noout
```

To verify the server's certificate is signed by your CA:

```
$ openssl verify -CAfile rootCA.pem <DOMAIN>.crt
```

When visiting a website over https, the browser basically runs the command above, to verify the certificate sent by the https server is signed by rootCA.pem.

Your tasks:

1. Make a directory for holding your CA certificate, signing request and keys.
2. Follow the tutorial above:
    a. Create a root CA (private) key
    b. Create a root CA certificate
    c. Create a certificate (private) key
    d. Create a certificate signing request
    e. Issue a signed certificate from the Certificate Signing Request
    f. Verify the issued certificate from step e) has been signed by your root CA

The exercise will return the following message:

```
$ openssl x509 -in <DOMAIN>.crt -text -noout

Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number:
            72:e8:63:24:d5:2e:fe:36:68:24:dc:4f:a3:e9:ed:84:2a:f9:57:74
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = DA, ST = Fyn, L = Odense, O = SDU, OU = MMMI, CN = localhost
        Validity
            Not Before: Oct  2 07:48:34 2019 GMT
            Not After : Jun 28 07:48:34 2022 GMT
        Subject: C = DA, ST = Fyn, L = Odense, O = SDU, OU = MMMI, CN =
localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b2:b6:33:38:c9:d4:e1:7f:72:c8:bd:93:22:ff:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:

keyid:19:6D:9E:C6:80:F9:A7:EC:7F:60:12:AD:46:73:D3:A8:9A:DC:55:55

            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment, Data
Encipherment
            X509v3 Subject Alternative Name:
                DNS:localhost
    Signature Algorithm: sha256WithRSAEncryption
         3e:04:54:f5:b5:f9:2e:29:31:9b:6d:09:c6:82:2b:e2:41:4d:
         ...

$ openssl verify -CAfile rootCA.pem <DOMAIN>.crt
<DOMAIN>.crt: OK
```

Exercise 02*

In this exercise the student has to push the two images from Lab 02 Exercise 2 and Exercise 4 to the Docker Hub registry.

When wanting to push an image to the Docker Hub registry, it is important to give the image a unique tag, that is specific for that image alone. Images can be tagged using the following command `$ docker tag <IMAGE> <TAG>`.
Tags normally follow the convention: `<USERNAME>/<IMAGENAME>`, like `leonlarsen/apacheserver`.

When an image has been tagged, the image can be pushed to Docker Hub using the command `$ docker push <TAG>`.

Remember to login to Docker Hub using `$ docker login`

Your tasks:

1. Tag the images in using the format mentioned above.
2. Login to Docker Hub.
3. Push the tagged image to Docker Hub.

The exercise will return the following message:

```
$ THE PUSH COMMAND

The push refers to repository [docker.io/<USERNAME>/<IMAGENAME>]
e98350dd688f: Mounted from <USERNAME>/<IMAGENAME>
10f1c6e9f6f1: Mounted from <USERNAME>/<IMAGENAME>
a4565839f6b5: Mounted from <USERNAME>/<IMAGENAME>
...
latest: digest:
sha256:3dfb3b2762095f360ae297d6dfedc8823a9d4cb26bc6ef7d886b262f875b3bc8 size:
3659
```

~~Exercise 03~~*

This exercise is deprecated. Solve it (at home) if you dare

In this exercise the student is going to create a Kubernetes cluster that sets up the host computer as a control-plane and connected computers as workers.

NOTE: When using a switch not connected to a DHCP server, you have to manually choose an IP-address. Look under your network settings -> (Advanced) ->IP settings -> Manual IP. Remember to have all nodes on the same subnet and use the subnet mask 255.255.255.0.

NOTE: The manager must run Linux, not MacOS or Windows.

A Kubernetes cluster is a bunch of computers connected to each other and managed through a control-plane with the purpose of distributing the workload onto a number of computers.
To initiate a Kubernetes cluster, three packages are needed on the control-plane, kubectl, kubelet and kubeadm. On the workers, only kubelet and kubeadm are needed.

The following guide walks through the kubectl installation process:

https://kubernetes.io/docs/tasks/tools/install-kubectl/

The following guide walks through the installation process:
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

Afterwards, the cluster has to be initialised. The following guide walks through the setup:
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

The command `$ kubectl get nodes` can be used to get an overview of the nodes currently in the Kubernetes cluster.

Your tasks:

1. Use the handed out switches and connect the groups computers to the same switch.
2. Install the kubectl, kubelet and kubeadm on the control-plane.
3. Install kubelet and kubeadm on the workers.
4. Initialise the cluster on the control-plane.
5. Join the cluster on all the workers.
6. Make sure that all the nodes are to be seen on the managers.

The exercise will return the following message:

```
$ kubectl get nodes

NAME                STATUS      ROLES          AGE             VERSION

tek-bro-stream01    Ready       master         14d             v1.18.6

tek-bro-stream02    Ready       <none>         13d             v1.18.6
```

Exercise 04*

To students who already have kubectl and a config:
Keep the current config and use `$ kubectl --kubeconfig=<KUBE CONFIG> get` .. etc.

In this exercise the student is going to install kubectl (Kube controller) on their machine. The kubectl, also referred to as the kubernetes CLI, lets you control your kubernetes clusters. This will be your main tool for using kubernetes, so getting familiar with this is very useful. By running the following syntax, you can make commands with kubectl

`$ kubectl <COMMAND> <RESOURCE TYPE> <NAME> <FLAGS>`

The following guide walks through the kubectl installation process:

https://kubernetes.io/docs/tasks/tools/install-kubectl/

The config should be placed in ~.kube folder found in the home directory. If it is not present, create the folder called ".kube". the config should be named "config" with no extension.

The command `$ kubectl get nodes` can be used to get an overview of the nodes currently in the Kubernetes cluster.

The command below is used to deploy something. Either a deployment, service etc.
`$ kubectl apply -f <PATH-TO-YAML>`

A command very similar to the one above is used to delete or remove the .yml deployment file again.
`$ kubectl delete -f <PATH-TO-YAML>`

The command below is used to get a resource, this can be either: deployments, pods, nodes, services, events.
`$ kubectl get <RESOURCE>`

This command was mentioned above but deserves some extra attention. It is a very useful command for problem solving. If a part of the deployment is not behaving as expected or not working at all, the events might give a clue as to what happened.
`$ kubectl get events`

This command is useful for getting a further description of a resource. For example about a node, pod, deployment or service. One should first tell kubectl what the resource is (deployment, node, pod etc.) followed by the name from the get command.
`$ kubectl describe <RESOURCE> <NAME>`

This command is used to forward a local port to a pod. This is useful because one can map a local port to a port in the pod. An example usage is when using a visualizer. Using this method only the local machine has access to the visualizer. If a service was used instead everyone would have access which is not as desirable.
`$ kubectl port-forward deployment/<NAME> <HOSTPORT>:<CONTAINERPORT>`

Your tasks:

1. Install kubectl following the guide provided
2. Make sure a config is installed by using `$ kubectl config view`
    a. If the config is not the right one use `--kubeconfig <KUBE-CONFIG>`
3. Play around with the commands. What resources are present on the kubernetes cluster currently?

The exercise will return something similar to the message below:

```
$ kubectl get nodes
NAME                STATUS     ROLES      AGE            VERSION
tek-bro-stream01    Ready      master     14d            v1.18.6
tek-bro-stream02    Ready      <none>     13d            v1.18.6
```

Exercise 05*

In this exercise the student is going to create a deployment called **spekt8** which displays all running Pods, Deployments, Services and Ingresses visually.

Pods

Pods are the smallest units of computing that you can create and manage in kubernetes. A pod is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers.
Click for more information

Deployments

A deployment is a controller used to describe which pods to run, and how many instances of the pod to create, by defining a number of replica sets. The deployment is defined in a yaml document, an example of this is shown below.
Click for more information

Services

A service is used to manage the reachability of the pods from both inside the cluster, and outside the cluster. A pod inside a cluster has a cluster ip, but to make sure that pods can be created and destroyed dynamically, the cluster defines a single DNS name for a set of pods, so you are able to reach the pod, from other pods, even though the cluster ip might change.
Click for more information

NodePort

The quote “Exposes the Service on each Node's IP at a static port” needs to be taken very literally. Read it a few times. Try and understand it and think about what it’s saying. “Exposes the service”, meaning that it will be open and available to connect to. But it is exposed on “each node’s IP”. Each and every node will point to the same service on the specified port. It doesn’t matter which node the service is running in, all nodes will point at the same service

from the same static port number. It is then possible to contact the service from outside the cluster using `<NodeIP>:<NodePort>`

Spekt8

The **spekt8** visualiser (https://github.com/spekt8/spekt8) is a visualiser made with the purpose of displaying the active Kubernetes Resources currently running on the Kubernetes Cluster.

The lines mentioned below need to be filled out. The student should choose a unique name or keyword to use and to recognise the deployment.
`metadata.name`
`spec.selector.matchLabels.component`
`spec.template.metadata.labels.component`

This is done by providing a NodeJS Express web server, showing Pods, Deployments, Services and Ingresses.

Your tasks:

1. Create a Kubernetes Deployment using kubectl to run the above-mentioned Visualiser. Use the provided manifest (yaml-file).
2. Port-forward port 3000 from the container to your hosts port 3000, to visit the Visualiser Dashboard.
    a. Using the command described in exercise 04
3. Access the Dashboard through localhost:3000 to see Kubernetes Resources.

IMPORTANT: There is an issue with the dashboard so there won't be anything visible, but that's alright.

Exercise 06*

In this exercise the student is going to create a Kubernetes Manifest (YAML-file) based on the compose file from Lab 02; Exercise 05. If you do not have a compose-file, use the one provided in resources.

Kubernetes Manifest

The manifest file(s) of kubernetes is where one defines kubernetes objects/resources. This could be deployments, services, namespaces etc.

Each type of object has a predetermined set of variables that has to be configured in order for kubernetes to manage the configuration.
Creating kubernetes resources can be done in two ways. One would be to create a manifest-file for each object desired to be deployed into the cluster, enhancing the readability of each manifest-file. Another way of doing it would be to define several resources within the same manifest-file. This can be done like in the example below, showcasing the definition of a deployment as the first resource with a service following below the '---' separation.

```
apiVersion: apps/v1                 # The apiVersion defining the the deployment
kind: Deployment                    # Defining Which Kubernetes Resource to use
metadata:                           # Metadata for the Deployment
  name: example-deployment          # Name of Deployment
spec:
  replicas: 3                       # Number of replicas
  selector:
    matchLabels:
      app: example-pod              # This label serves as a reference to pod(s)
  template:                         # Template for the Pod(s) to run
    metadata:                       # Metadata for the Pod(s)
      labels:
        app: example-pod            # Label matching the matchLabel.
    spec:
      containers:
      - name: hello-world           # Name of the Pods
        image: hello-world          # Image running in the Pods
        command: ["/bin/echo"]      # Command to run
        args: ["Message"]           # Command arguments (and options)
        env:                        # List of environment variables
        - name: ENV_NAME
          value: "Env value"
        ports:
        - containerPort: 8080       # Port to expose
---                                 # This '---'  serves as the separation.
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  type: NodePort                    # Type of the service
  ports:
  - port: 80                        # Cluster IP port to map to
    targetPort: 8080                # Containerport to map from
    nodePort: 32323                 # Mapping to all nodes on port [30000:32767]
  selector:
    app: example-pod                # This label serves as a reference to pod(s)
```

Use the spekt8 visualiser created in the previous exercise to see the running resources.

```
$ kubectl apply -f <PATH-TO-YAML>

deployment.apps/l2e5 created
service/l2e5 created
```

Your tasks:

1. Make a Kubernetes Manifest that creates a Deployment containing the Docker services from Lab 02; Exercise 05.
2. Visit the spekt8 visualiser to see the running resources (might not work).
3. Visit the node03.stream.stud-srv.sdu.dk:<NODEPORT> and test that the system is functional.

Exercise 07

In this exercise the students are going to discuss the topic regarding distributed systems and Kubernetes

Your tasks:

1. What is a control-plane in Kubernetes and what does it do?
2. What is a worker in Kubernetes and what does it do?
3. What are the benefits of Kubernetes?
4. What are the disadvantages of Kubernetes?