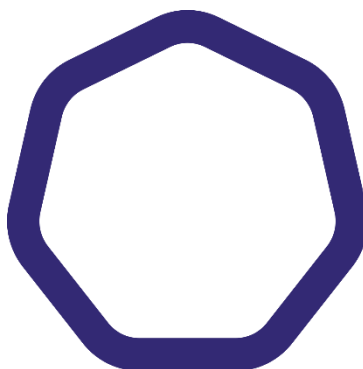


Министерство цифрового развития, связи и массовых коммуникаций  
Российской Федерации  
Ордена Трудового Красного Знамени федеральное государственное  
бюджетное образовательное учреждение высшего образования  
«Московский технический университет связи и информатики»  
(МТУСИ)



Кафедра: Сетевые информационные технологии и сервисы

Отчёт по лабораторной работе №5  
«Изучение приложения»

Выполнил:  
Студенты 1 курса  
Группы М092401(75)  
Цыганков Р.О.  
Проверил:  
к.т.н., Фатхулин Т. Д.

## **Задание**

Познакомиться как работают поды и узлы в Kubernetes и познакомиться с диагностикой развёрнутых приложений на Kubernetes.

### **Выполнение работы**

Для выполнения лабораторных работ необходимо:

- Зайти на сайт: <https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/explore/explore-intro/>;
- Изучить теоретическую часть;
- Нажать на кнопку «Начать интерактивный урок».

#### **Теоретическая часть работы:**

##### **Поды Kubernetes**

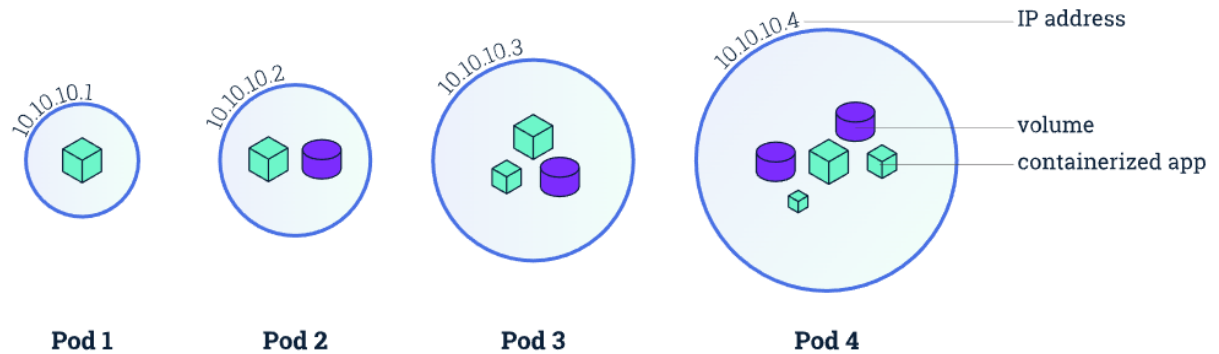
Под — это абстрактный объект Kubernetes, представляющий собой группу из одного или нескольких контейнеров приложения (например, Docker или rkt) и совместно используемых ресурсов для этих контейнеров. Ресурсами могут быть:

- Общее хранилище (тома);
- Сеть (уникальный IP-адрес кластера);
- Информация по выполнению каждого контейнера (версия образа контейнера или используемые номера портов).

Под представляет специфичный для приложения "логический хост" и может содержать разные контейнеры приложений, которые в общем и целом тесно связаны. Например, в поде может размещаться как контейнер с приложением на Node.js, так и другой контейнер, который использует данные от веб-сервера Node.js. Все контейнеры в поде имеют одни и те же IP-адрес и пространство порта, выполняющиеся в общем контексте на одном и том же узле.

Поды — неделимая единица в платформе Kubernetes. При создании развёртывания в Kubernetes, создаются поды с контейнерами внутри (в отличие от непосредственного создания контейнеров). Каждый Pod-объект связан с узлом, на котором он размещён, и остаётся там до окончания работы (согласно стратегии перезапуска) либо удаления. В случае неисправности узла такой же под будет распределён на другие доступные узлы в кластере.

## Схема подов:



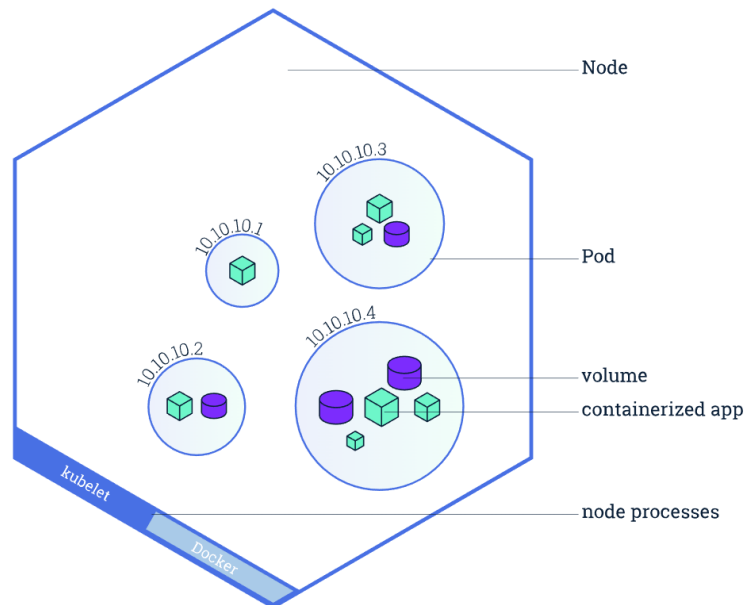
## Узлы

Под всегда работает в узле. Узел — это рабочая машина в Kubernetes, которая в зависимости от кластера может быть либо виртуальной, либо физической. Каждый узел управляется мастером (ведущим узлом). Узел может содержать несколько подов, которые мастер Kubernetes автоматически размещает на разные узлы кластера. Ведущий узел при автоматическом планировании (распределении подов по узлам) учитывает доступные ресурсы на каждом узле.

В каждом узле Kubernetes как минимум работает:

- Kubelet — процесс, отвечающий за взаимодействие между мастером Kubernetes и узлом; он управляет подами и запущенными контейнерами на рабочей машине;
- Среда выполнения контейнера (например, Docker или rkt), отвечающая за получение (загрузку) образа контейнера из реестра, распаковку контейнера и запуск приложения.

Схема узла:



### Диагностика с помощью kubectl

Наиболее распространенные операции для диагностики выполняются с использованием следующих команд kubectl:

- kubectl get — вывод списка ресурсов;
- kubectl describe — вывод подробной информации о ресурсе;
- kubectl logs — вывод логов контейнера в поде;
- kubectl exec — выполнение команды в контейнере пода;

Перечисленные выше команды можно использовать, чтобы узнать, когда и где приложения были развернуты, их текущее состояние и конфигурацию.

## Практическая часть работы:

### 1. Проверка конфигурации приложения:

< | Module 3 - Explore your app

Step 1 of 4 ▶

### Step 1 Check application configuration

Let's verify that the application we deployed in the previous scenario is running. We'll use the `kubectl get` command and look for existing Pods:

```
kubectl get pods ✓
```

If no pods are running then it means the interactive environment is still reloading it's previous state. Please wait a couple of seconds and list the Pods again. You can continue once you see the one Pod running.

Next, to view what containers are inside that Pod and what images are used to build those containers we run the `describe pods` command:

```
kubectl describe pods ✓
```

Terminal

```
Ready:      True
Restart Count: 0
Environment: <none>
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-td748 (ro)
Conditions:
  Type              Status
  Initialized        True
  Ready              True
  ContainersReady    True
  PodScheduled       True
Volumes:
  default-token-td748:
    Type:          Secret (a volume populated by a Secret)
    SecretName:     default-token-td748
    Optional:       false
QoS Class:         BestEffort
Node-Selectors:    <none>
Tolerations:       node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                   node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason      Age   From              Message
  ----    -
  Normal  Scheduled   14m   default-scheduler Successfully assigned default/kubernetes-bootcamp-fb5c67579-swfvq to minikube
  Normal  Pulled      14m   kubelet           Container image "gcr.io/google-samples/kubern
etes-bootcamp:v1" already present on machine
  Normal  Created     14m   kubelet           Created container kubernetes-bootcamp
  Normal  Started     14m   kubelet           Started container kubernetes-bootcamp
$
```

### 2. Просмотр приложения в терминале:

< | Module 3 - Explore your app

Step 2 of 4 ▶

### Step 2 Show the app in the terminal

Recall that Pods are running in an isolated, private network - so we need to proxy access to them so we can debug and interact with them. To do this, we'll use the `kubectl proxy` command to run a proxy in a second terminal window. Click on the command below to automatically open a new terminal and run the `proxy`:

```
echo -e "\n\n[e]Starting Proxy. After starting it will not output a response. Please click the first Terminal Tab\n"; kubectl proxy ✓
```

Now again, we'll get the Pod name and query that pod directly through the proxy. To get the Pod name and store it in the `POD_NAME` environment

Terminal

Terminal 2

```
default-token-td748:
  Type:          Secret (a volume populated by a Secret)
  SecretName:     default-token-td748
  Optional:       false
QoS Class:         BestEffort
Node-Selectors:    <none>
Tolerations:       node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                   node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason      Age   From              Message
  ----    -
  Normal  Scheduled   14m   default-scheduler Successfully assigned default/kubernetes-bootcamp-fb5c67579-swfvq to minikube
  Normal  Pulled      14m   kubelet           Container image "gcr.io/google-samples/kubern
etes-bootcamp:v1" already present on machine
  Normal  Created     14m   kubelet           Created container kubernetes-bootcamp
  Normal  Started     14m   kubelet           Started container kubernetes-bootcamp
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-fb5c67579-swfvq
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
curl: (7) Failed to connect to localhost port 8001: Connection refused
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-fb5c67579-swfvq
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-fb5c67579-swfvq | v=1
$
```

### 3. Просмотр логов контейнера:

< | Module 3 - Explore your app

◀ Step 3 of 4 ▶

#### Step 3 View the container logs

Anything that the application would normally send to `STDOUT` becomes logs for the container within the Pod. We can retrieve these logs using the `kubectl logs` command:

```
kubectl logs $POD_NAME ✓
```

*Note: We don't need to specify the container name, because we only have one container inside the pod.*

CONTINUE

Terminal

Terminal 2 +

```
camp-fb5c67579-swfvg to minikube
Normal Pulled 14m kubelet Container image "gcr.io/google-samples/kubern
etes-bootcamp:v1" already present on machine
Normal Created 14m kubelet Created container kubernetes-bootcamp
Normal Started 14m kubelet Started container kubernetes-bootcamp
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata
.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-fb5c67579-swfvg
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
curl: (7) Failed to connect to localhost port 8001: Connection refused
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata
.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-fb5c67579-swfvg
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-fb5c67579-swfvg | v=1
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2022-12-10T15:25:00.679Z | Running On: kubernetes-boot
camp-fb5c67579-swfvg

Running On: kubernetes-bootcamp-fb5c67579-swfvg | Total Requests: 1 | App Uptime: 944.439 s
econds | Log Time: 2022-12-10T15:40:45.118Z
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2022-12-10T15:25:00.679Z | Running On: kubernetes-boot
camp-fb5c67579-swfvg

Running On: kubernetes-bootcamp-fb5c67579-swfvg | Total Requests: 1 | App Uptime: 944.439 s
econds | Log Time: 2022-12-10T15:40:45.118Z
$
```

### 4. Выполнение команд в контейнере:

< | Module 3 - Explore your app

◀ Step 4 of 4 ▶

#### Step 4 Executing command on the container

We can execute commands directly on the container once the Pod is up and running. For this, we use the `exec` command and use the name of the Pod as a parameter. Let's list the environment variables:

```
kubectl exec $POD_NAME -- env ✓
```

Again, worth mentioning that the name of the container itself can be omitted since we only have a single container in the Pod.

Next let's start a bash session in the Pod's container:

```
kubectl exec -ti $POD_NAME -- bash ✓
```

Terminal

Terminal 2 +

```
NODE_VERSION=6.3.1
HOME=/root
$ kubectl exec -ti $POD_NAME -- bash
root@kubernetes-bootcamp-fb5c67579-swfvg:/# cat server.js
var http = require('http');
var requests=0;
var podname= process.env.HOSTNAME;
var startTime;
var host;
var handleRequest = function(request, response) {
  response.setHeader('Content-Type', 'text/plain');
  response.writeHead(200);
  response.write("Hello Kubernetes bootcamp! | Running on: ");
  response.write(host);
  response.end(" | v=1\n");
  console.log("Running On:", host, "| Total Requests:", ++requests, "| App Uptime:", (new Date() - startTime)/1000, "seconds", "| Log Time:", new Date());
}
var www = http.createServer(handleRequest);
www.listen(8080,function () {
  startTime = new Date();
  host = process.env.HOSTNAME;
  console.log ("Kubernetes Bootcamp App Started At:",startTime, "| Running On: ",host, "\n" );
});
root@kubernetes-bootcamp-fb5c67579-swfvg:/# curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-fb5c67579-swfvg | v=1
root@kubernetes-bootcamp-fb5c67579-swfvg:/# exit
exit
$
```

## Заключение

В результате работы мы получили практические навыки работы с подами и узлами в Kubernetes и с диагностикой развёрнутых приложений на Kubernetes.