



3

## ЛР 3. Kubernetes

### ▼ Задача

Установить Kubernetes на локальную машину (приведен пример для minikube на Windows 10/11). Развернуть тестовый сервис

### ▼ Описание

#### ▼ Часть 1. Установка minikube (опционально)

1. Установить и запустить [Docker Desktop](#), убедиться что работает
2. Скачать [kubectl.exe](#), утилиту для работы с Kubernetes CLI
3. Настроить среду для работы: запустить удобный терминал (*wincmd, Powershell, bash...*), добавить `kubectl.exe` в окружение `PATH`, либо просто держать терминал открытым в папке, где лежит `kubectl` (расширение `exe` желательно убрать)

#### ▼ Пример

```
llidd@winlid MINGW64 ~
$ ls -la | grep kubectl
-rwxr-xr-x 1 llidd 197121 47827456 Dec 16 2021 kubectl
```

4. Скачать с GitHub и запустить [minikube-installer.exe](#)

5. Проверить, что все успешно установилось — командой `minikube version`

6. Запустить minikube

▼ `minikube start`

```
llidd@winlid MINGW64 ~/containers
$ minikube version
minikube version: v1.33.1
commit: 5883c09216182566a63dff4c326a6fc9ed2982ff

llidd@winlid MINGW64 ~/containers
$ minikube start
◆ minikube v1.33.1 на Microsoft Windows 11 Pro 10.0.22631.3447 Build 22631.3447
◆ Automatically selected the docker driver. Other choices: virtualbox, ssh
◆ Using Docker Desktop driver with root privileges
◆ Starting "minikube" primary control-plane node in "minikube" cluster
◆ Pulling base image v0.0.44 ...
◆ Скачивается Kubernetes v1.30.0 ...
    > preloaded-images-k8s-v18-v1...: 342.90 MiB / 342.90 MiB 100.00% 17.71 Mi
    > gcr.io/k8s-minikube/kicbase...: 481.58 MiB / 481.58 MiB 100.00% 7.14 Mi
◆ Creating docker container (CPUs=2, Memory=1966MB) ...
◆ Подготавливается Kubernetes v1.30.0 на Docker 26.1.1 ...
    • Generating certificates and keys ...
    • Booting up control plane ...
    • Configuring RBAC rules ...
◆ Configuring bridge CNI (Container Networking Interface) ...
◆ Компоненты Kubernetes проверяются ...
    • Используется образ gcr.io/k8s-minikube/storage-provisioner:v5
◆ Включенные дополнения: storage-provisioner, default-storageclass
◆ Готово! kubectl настроен для использования кластера "minikube" и "default" пространства имен по умолчанию
```

7. Проверить, что все успешно запустилось с помощью команд `docker ps` (в списке должен быть контейнер с именем *minikube*) и `kubectl config view` (должен отобразиться конфиг созданного кластера)

▼ Пример

```

- context:
  cluster: minikube
  extensions:
  - extension:
    last-update: Wed, 15 May 2024 00:58:29 MSK
    provider: minikube.sigs.k8s.io
    version: v1.33.1
    name: context_info
  namespace: default
  user: minikube
name: minikube

```

```

llidd@winlid MINGW64 ~/containers
$ docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED             STATUS              NAMES
TUS           gcr.io/k8s-minikube/kicbase:v0.0.44   "/usr/local/bin/entr..."   9 minutes ago   Up 9 minutes   127.0.0.1:63100->22/tcp, 127.0.0.1:63101->2376/tcp, 127.0.0.1:63098->5000/tcp, 127.0.0.1:63099->8443/tcp, 127.0.0.1:63097->32443/tcp   minikube

```

- Порадоваться, что в настоящее время установить и пощупать кубер это довольно несложно и быстро :)

## ▼ Часть 2. Создаем объекты через CLI

В качестве базового примера развернем сервис Nextcloud на БД PostgreSQL

Шаг 1. В терминале [в текущей папке] создать yaml-файлы (манифесты) конфигмапы, сервиса и деплоимента (**не пропустить отступы, в YAML-файлах это критично**). Значения для пароля изменить на свое, остальные по желанию

### ▼ pg\_configmap.yaml

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-configmap
  labels:
    app: postgres
data:
  POSTGRES_DB: "postgres"

```

```
POSTGRES_USER: "postgres"  
POSTGRES_PASSWORD: "any_password_u_want"
```

▼ pg\_service.yml

```
apiVersion: v1  
kind: Service  
metadata:  
  name: postgres-service  
  labels:  
    app: postgres  
spec:  
  type: NodePort  
  ports:  
    - port: 5432  
  selector:  
    app: postgres
```

▼ pg\_deployment.yml

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: postgres  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: postgres  
  template:  
    metadata:  
      labels:  
        app: postgres  
    spec:  
      containers:
```

```
- name: postgres-container
  image: postgres:14
  resources:
    limits:
      cpu: 200m
      memory: 256Mi
    requests:
      cpu: 100m
      memory: 128Mi
  imagePullPolicy: "IfNotPresent"
  ports:
    - containerPort: 5432
  envFrom:
    - configMapRef:
        name: postgres-configmap
```

Шаг 2. Прогнать манифесты, чтобы описанные объекты создались в кластере, с помощью команды `kubectl create -f имя_файла`

▼ Пример

```
llidd@winlid MINGW64 ~/containers
$ kubectl config current-context
minikube

llidd@winlid MINGW64 ~/containers
$ kubectl create -f pg_configmap.yml
configmap/postgres-configmap created

llidd@winlid MINGW64 ~/containers
$ kubectl create -f pg_service.yml
service/postgres-service created

llidd@winlid MINGW64 ~/containers
$ kubectl create -f pg_deployment.yml
deployment.apps/postgres-deployment created
```



Вопрос: важен ли порядок выполнения этих манифестов?  
Почему?

Шаг 3. Проверить, что все ресурсы успешно создались, с помощью команды `kubectl get тип_ресурса`. Чтобы посмотреть конкретный ресурс, нужно обратиться к нему через `kubectl describe тип_ресурса/имя_ресурса` (вместо слеша можно пробел, такой синтаксис тоже поддерживается). Стоит также обратить внимание, что атрибутов при этом будет несколько больше, чем в исходном yaml-файле — это объясняется тем, что манифест это лишь шаблон для будущего ресурса, скелет, остальные настройки “дописывает” сам кубер

#### ▼ Пример

```
llidd@winlid MINGW64 ~/containers
$ kubectl get configmap
NAME          DATA   AGE
kube-root-ca.crt   1    15m
postgres-configmap 3    106s

llidd@winlid MINGW64 ~/containers
$ kubectl get service
NAME           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1      <none>          443/TCP       15m
postgres-service  NodePort   10.103.32.249  <none>          5432:30702/TCP 108s

llidd@winlid MINGW64 ~/containers
$ kubectl describe service/postgres-service
Name:           postgres-service
Namespace:      default
Labels:         app=postgres
Annotations:   <none>
Selector:       app=postgres
Type:          NodePort
IP Family Policy: SingleStack
IP Families:   IPv4
IP:             10.103.32.249
IPs:            10.103.32.249
Port:          <unset>  5432/TCP
TargetPort:    5432/TCP
NodePort:      <unset>  30702/TCP
Endpoints:     10.244.0.3:5432
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
```

Шаг 4. Создать `nextcloud.yml` и повторить для него шаги 1-3. Чтобы создать объекты в кубере с помощью манифестов, необязательно делить их на отдельные yaml-файлы, можно сделать один большой, с разделителем, как в примере ниже. Значения для юзера и пароля изменить на свое, остальные по желанию

▼ `nextcloud.yml`

```
apiVersion: v1
kind: Secret
metadata:
  name: nextcloud-secret
  labels:
    app: nextcloud
type: Opaque
stringData:
  NEXTCLOUD_ADMIN_PASSWORD: "literally_any_password"
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nextcloud
  labels:
    app: nextcloud
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nextcloud
  template:
    metadata:
      labels:
        app: nextcloud
  spec:
    containers:
      - name: nextcloud
```

```
image: docker.io/nextcloud:stable-apache
resources:
  limits:
    cpu: 500m
    memory: 256Mi
  requests:
    cpu: 250m
    memory: 128Mi
ports:
- name: http
  containerPort: 80
  protocol: TCP
env:
- name: NEXTCLOUD_UPDATE
  value: '1'
- name: ALLOW_EMPTY_PASSWORD
  value: 'yes'
- name: POSTGRES_HOST
  value: postgres-service
- name: POSTGRES_DB
  valueFrom:
    configMapKeyRef:
      name: postgres-configmap
      key: POSTGRES_DB
- name: NEXTCLOUD_TRUSTED_DOMAINS
  value: "127.0.0.1"
- name: POSTGRES_USER
  valueFrom:
    configMapKeyRef:
      name: postgres-configmap
      key: POSTGRES_USER
- name: POSTGRES_PASSWORD
  valueFrom:
    configMapKeyRef:
      name: postgres-configmap
      key: POSTGRES_PASSWORD
```

```
- name: NEXTCLOUD_ADMIN_USER
  value: any_name_you_want
- name: NEXTCLOUD_ADMIN_PASSWORD
  valueFrom:
    secretKeyRef:
      name: nextcloud-secret
      key: NEXTCLOUD_ADMIN_PASSWORD
  imagePullPolicy: IfNotPresent
  restartPolicy: Always
  dnsPolicy: ClusterFirst
```

▼ Можно обратить внимание, что при попытке посмотреть секрет, содержимое не показывается:

```
llidd@winlid MINGW64 ~/containers
$ kubectl create -f nextcloud.yml
secret/nextcloud-secret created
deployment.apps/nextcloud created

llidd@winlid MINGW64 ~/containers
$ kubectl describe secret/nextcloud-secret
Name:           nextcloud-secret
Namespace:      default
Labels:         app=nextcloud
Annotations:    <none>

Type:  Opaque

Data
====
NEXTCLOUD_ADMIN_PASSWORD:  5 bytes
```

Шаг 5. После успешного запуска пода можно проверить его состояние с помощью команды `kubectl logs имя_пода`. Nextcloud начнет установку (самоинициализацию) автоматически, какое-то время следует подождать до ее ~~скорее всего~~ успешного завершения

▼ Должно получиться что-то такое:

```
llidd@winlid MINGW64 ~/containers
$ kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
nextcloud-d7fb4787-dxq8r   1/1     Running   0          5s
postgres-7d4c95fdf8-n7znz 1/1     Running   0          99s

llidd@winlid MINGW64 ~/containers
$ kubectl logs nextcloud-d7fb4787-dxq8r
Initializing nextcloud 28.0.5.1 ...
New nextcloud instance
Installing with PostgreSQL database
=> Searching for scripts (*.sh) to run, located in the folder: /docker-entrypoint-hooks.d/pre-installation
Starting nextcloud installation

llidd@winlid MINGW64 ~/containers
$ kubectl logs nextcloud-d7fb4787-dxq8r
Initializing nextcloud 28.0.5.1 ...
New nextcloud instance
Installing with PostgreSQL database
=> Searching for scripts (*.sh) to run, located in the folder: /docker-entrypoint-hooks.d/pre-installation
Starting nextcloud installation
Nextcloud was successfully installed
Setting trusted domains...
System config value trusted_domains => 1 set to string 127.0.0.1
=> Searching for scripts (*.sh) to run, located in the folder: /docker-entrypoint-hooks.d/post-installation
Initializing finished
```

## ▼ Часть 3. Подключение извне (опционально)

В предыдущих частях была успешна создана БД Postgresql, к которой подключается сервис Nextcloud. Чтобы начать им пользоваться, необходимо осуществить доступ к самому интерфейсу сервиса, для этого обычно используется Ingress. К сожалению, работа с ингрессами на локальных экземплярах minikube [особенно на порте 80] затруднена, поэтому будет использоваться встроенный функционал minikube, конкретно Service + NodePort

Шаг 1. Кроме уже упомянутого способа выше (через манифест), объект типа Service для конкретного деплоимента можно создать специальной командой `kubectl expose deployment имя_деплоимента параметры_перенаправления`, которая автоматически обеспечит нужное перенаправление портов. При локальном использовании minikube удобнее всего работать с NodePort, т.е. форвардинг трафика через саму ноду minikube. Сама команда:

▼ `kubectl expose deployment nextcloud --type=NodePort --port=80`

```
llidd@winlid MINGW64 ~/containers
$ kubectl expose deployment nextcloud --type=NodePort --port=80
service/nextcloud exposed

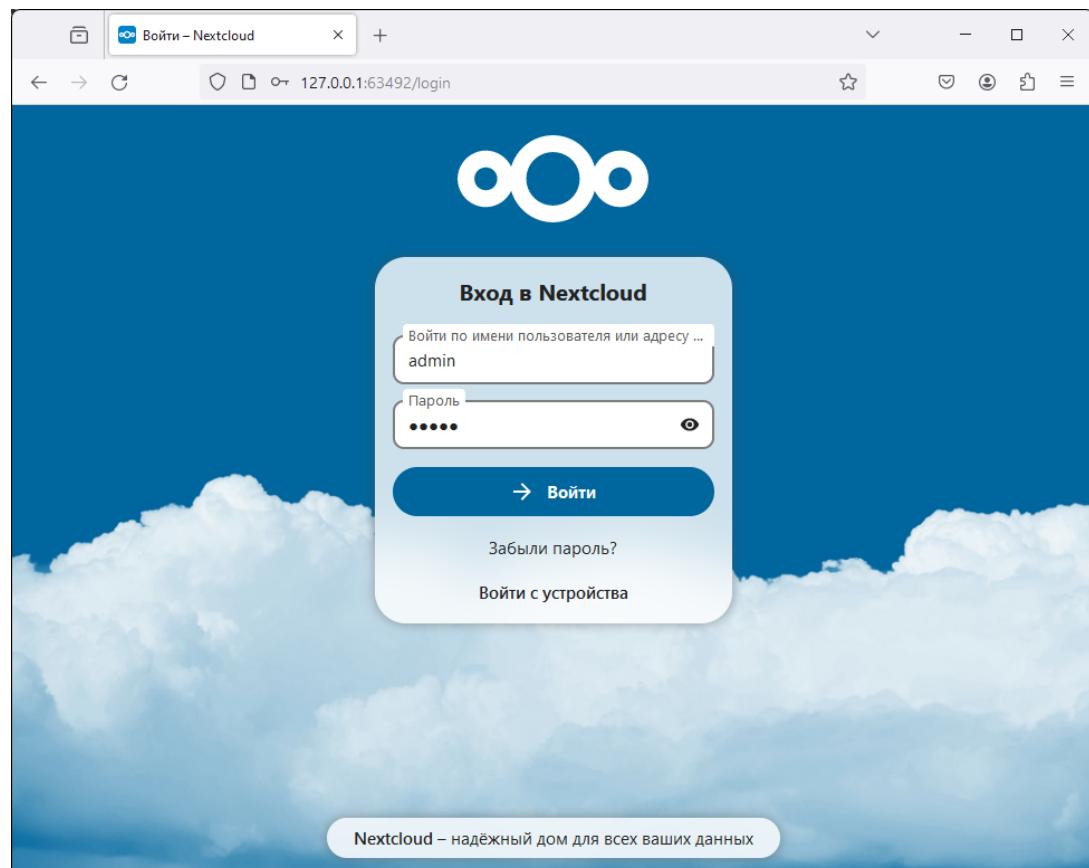
llidd@winlid MINGW64 ~/containers
$ kubectl describe service/nextcloud
Name:           nextcloud
Namespace:      default
Labels:          app=nextcloud
Annotations:    <none>
Selector:       app=nextcloud
Type:           NodePort
IP Family Policy: SingleStack
IP Families:   IPv4
IP:             10.96.203.143
IPs:            10.96.203.143
Port:           <unset>  80/TCP
TargetPort:     80/TCP
NodePort:       <unset>  30273/TCP
Endpoints:     10.244.0.8:80
Session Affinity: None
External Traffic Policy: Cluster
Events:         <none>
```

Шаг 2. Осуществить туннелирование трафика между нодой minikube и Сервисом с помощью команды `minikube service имя_сервиса`. Команда "захватит" терминал на все время выполнения (в данном случае до ручного прерывания по *Ctrl-C*)

Если все предыдущие шаги выполнены правильно и нигде не было ошибок, то после выполнения команды должно открыться вкладка в браузере с сервисом Nextcloud. Чтобы залогиниться, надо ввести логин и пароль, указанные в манифесте ранее (переменные `NEXTCLOUD_ADMIN_USER, NEXTCLOUD_ADMIN_PASSWORD`). Сама команда:

▼ `minikube service nextcloud`

```
llidd@winlid MINGW64 ~/containers
$ minikube service nextcloud
|-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | nextcloud | 80 | http://192.168.49.2:30273 |
|-----|-----|-----|-----|
◆ Starting tunnel for service nextcloud.
|-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | nextcloud | | http://127.0.0.1:63492 |
|-----|-----|-----|-----|
◆ Opening service default/nextcloud in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```



**На этом установку полноценного веб-сервиса с нуля можно считать успешно завершённой :)**

\*Шаг 3. Установить допкомпонент dashboard для minikube, команда `minikube dashboard --url`. Полученный url вставить в адресную строку в

браузере, если не откроется само (выполненная команда, вероятно, "захватит" терминал, но он больше и не понадобится). Можно походить по страницам, посмотреть на все ранее созданные объекты уже на дашборде, удалить, создать новые, отредактировать

### ▼ Пример

The screenshot shows the Kubernetes Dashboard interface. On the left, a sidebar lists various resources: CronJobs, DaemonSets, Deployments, Jobs, Pods, ReplicaSets, ReplicationControllers, and StatefulSets. Under the 'Service' section, Ingresses, Ingress Classes, Services, Config Maps, Persistent Volume Claims, Secrets, and Storage Classes are listed. The 'Cluster' section includes Cluster Role Bindings, Cluster Roles, Events, Namespaces, Network Policies, Nodes, Persistent Volumes, Role Bindings, Roles, Service Accounts, and Custom Resource Definitions. The 'Settings' and 'About' sections are also present.

**Workload Status:**

- Deployments: Running 2
- Pods: Running 2
- Replica Sets: Running 2

**Deployments:**

Name	Images	Labels	Pods	Created
nextcloud	docker.io/nextcloud/stable-apache	app:nextcloud	1 / 1	8 minutes ago
postgres	postgres:14	-	1 / 1	10 minutes ago

**Pods:**

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
nextcloud-7fb4787-depl	docker.io/nextcloud/stable-apache	app:nextcloud pod-template-hash: d7fb4787	minikube	Running	0	-	-	8 minutes ago
postgres-7d4c95f97-zoz	postgres:14	app:postgres pod-template-hash: 7d4c95f97	minikube	Running	0	-	-	10 minutes ago

**Replica Sets:**

Name	Images	Labels	Pods	Created
nextcloud-7fb4787	docker.io/nextcloud/stable-apache	app:nextcloud pod-template-hash: d7fb4787	1 / 1	8 minutes ago
postgres-7d4c95f97	postgres:14	app:postgres pod-template-hash: 7d4c95f97	1 / 1	10 minutes ago



Вопрос: что (и почему) произойдет, если отскейлить количество реплик postgres-deployment в 0, затем обратно в 1, после чего попробовать снова зайти на Nextcloud?

### ▼ Задание

Осуществить манипуляции над манифестами из примера, чтобы получить следующее:

- Для постгреса перенести `POSTGRES_USER` И `POSTGRES_PASSWORD` из конфигмапы в секреты (очевидно, понадобится новый манифест для сущности `Secret`)

- Для некстклауда перенести его переменные (`NEXTCLOUD_UPDATE`, `ALLOW_EMPTY_PASSWORD` и проч.) из деплоимента в конфигмапу (очевидно, понадобится новый манифест для сущности ConfigMap)
- Для некстклауда добавить `Liveness` и `Readiness` пробы

## ▼ Отчетность

Ссылка на git-репозиторий от команды, где содержится

- Все нужные манифесты
- README.md с описанием хода работы и скриншотами
- Ответы на доп. вопросы