Отчет 20 k8s

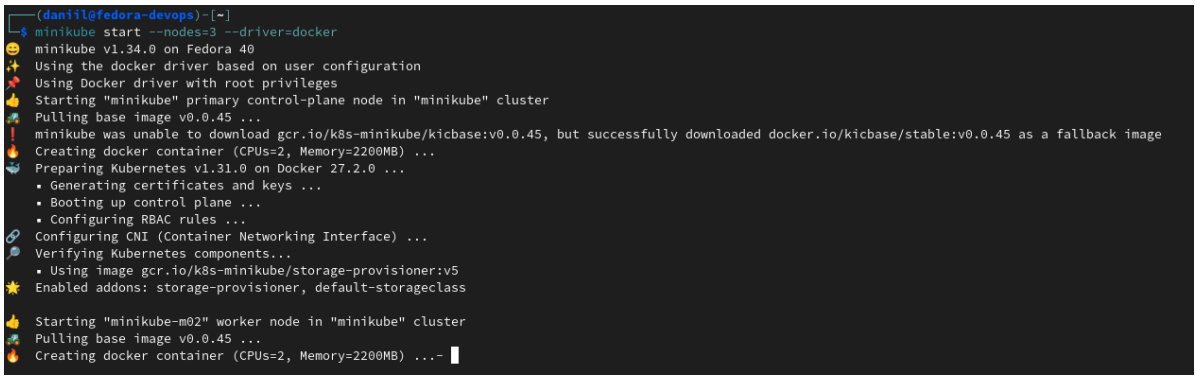
Кластер minikube на будущее

Так как в дальнейших лабах нужно минимум 2 воркер ноды, то сразу решил поднять локально кластер на несколько нод через minikube (развертка на него “лучше гуглилась”).

1. Запустим minikube с несколькими узлами `minikube start --nodes=3 --driver=docker`:

   `-nodes=3` создаст один мастер-узел и два рабочих узл

   `-driver=docker` запустит minikube внутри docker контейнеров



```
┌──(daniil@fedora-devops)-[~]
└─$ minikube start --nodes=3 --driver=docker
😄  minikube v1.34.0 on Fedora 40
✨  Using the docker driver based on user configuration
📌  Using Docker driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
❗  minikube was unable to download gcr.io/k8s-minikube/kicbase:v0.0.45, but successfully downloaded docker.io/kicbase/stable:v0.0.45 as a fallback image
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass

👍  Starting "minikube-m02" worker node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.49.2
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
    ▪ env NO_PROXY=192.168.49.2
🔎  Verifying Kubernetes components...

👍  Starting "minikube-m03" worker node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.49.2,192.168.49.3
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
    ▪ env NO_PROXY=192.168.49.2
    ▪ env NO_PROXY=192.168.49.2,192.168.49.3
🔎  Verifying Kubernetes components...
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
┌──(daniil@fedora-devops)-[~]
```

2. Убедимся, что все работает

   Выполним **просмотр доступных кластеров и выберем нужный** через

   `kubectl config get-context` - все доступные кластеры

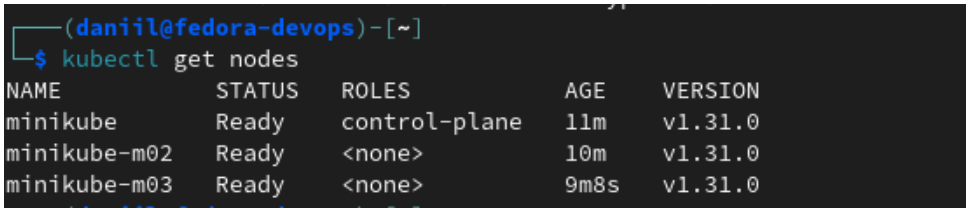
   `kubectl config current-context` - получения используемого кластера

   `kubectl config use-context minikube` - использовать определенный кластер (наверно надо было указать имя кластера при создании, но этот момент я чот упустил)

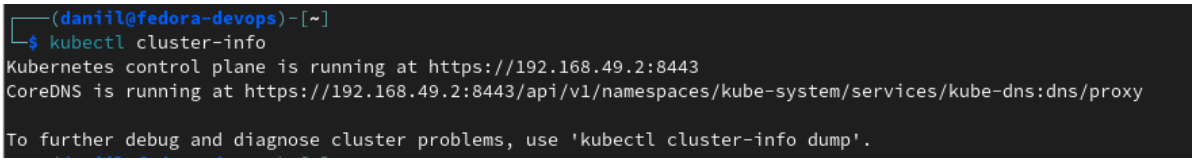
```
┌──(daniil@fedora-devops)-[~]
└─$ kubectl get context
error: the server doesn't have a resource type "context"
┌──(daniil@fedora-devops)-[~]
└─$ kubectl config current-context
minikube
┌──(daniil@fedora-devops)-[~]
└─$ kubectl config use-context minikube
Switched to context "minikube".
┌──(daniil@fedora-devops)-[~]
└─$
```

Также выведем **доступные ноды** в кластере `kubectl get nodes`

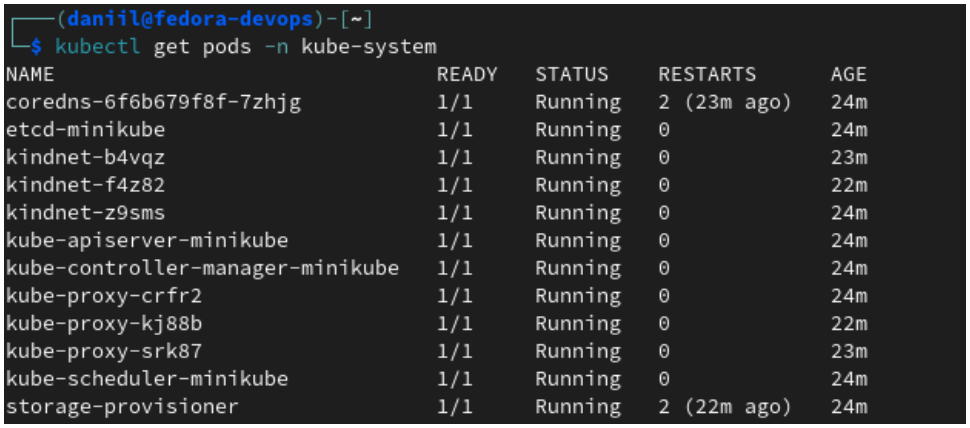
Видим три узла - один мастер и два рабочих



`kubectl cluster-info` выведем информацию об кластере



`kubectl get pods -n kube-system` выведем информацию об подах всистемном (основном) namespace нашего control plane



3. Установка сетевой подсетки

   Для работы подов в кластере требуется сетевая подсеть. Например, можно использовать Calico:

```
kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>
```

Выполнение лабораторной

Для начала создадим namespace `kubectl create namespace dev`

mysql

mysql-statefulset

Создадим путем проб и ошибок следующий манифест Stateful объект рабочей нагрузки для развертывания mysql. Возможно он слишком “большой”, мне кажется, что в рамках работы мог быть меньше, но старался использовать как можно больше штук, чтобы еще и не поломалось, для “набивания руки”, а то будет как с ansible.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-set
  namespace: dev
spec:
  serviceName: "mysql-service" # имя сервиса, на который буду
т ссылаться поды
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
```

```
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.4.3
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: rootpassword
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
```

Шпаргалка для себя (можно не читать):

1. **apiVersion: apps/v1**
   Указывает на версию api, которую использует kubernetes для создания этого объекта. Версия
   apps/v1, которая поддерживает такие объекты, как **Deployment**, **StatefulSet и тд.**
2. **kind: StatefulSet**

   В kind указывается тип создаваемого объекта. Еще раз скопирую про StatefulSet **-** это объект Kubernetes, который управляет развертыванием и масштабированием подов с сохранением состояния. Он используется для приложений, которым необходимо иметь стабильные идентификаторы сети, стабильное хранилище, и которые могут нуждаться в упорядоченном развертывании и удалении.
3. **metadata**

   В разделе `metadata` указываются основные метаданные объекта:

1. **name** - указывает имя для StatefulSet (в данном случае **mysql-set**). Это имя будет использоваться для **идентификации этого объекта в кластере**.
2. **namespace** - указывает пространство имён (namespace), в котором будет создан объект. В данном случае это **dev**, что означает, что StatefulSet будет создан в пространстве имен с именем **dev**.

4. **spec**

   Раздел `spec` описывает желаемое состояние объекта, а именно, как должен быть развернут StatefulSet. Из чего состоит:

   1. **serviceName** - указывает имя сервиса, который будет связан с этим StatefulSet. В данном случае это `"mysql-service"`. Это имя сервиса, через который поды StatefulSet будут доступны для других приложений или компонентов в кластере.
   2. **replicas** - указывает количество реплик (подов), которые должны быть развернуты. В данном случае указывается `1`, что означает, что будет развернут только один под с mysql.
   3. **selector -** определяет, как Kubernetes будет определять, какие поды принадлежат данному StatefulSet. В данном случае используется лейбл `app: mysql` для выбора подов. Под должен иметь этот лейбл, чтобы считаться частью этого StatefulSet.
   4. **template** - в этом разделе описан шаблон пода, который будет развернут в StatefulSet. Он включает следующие элементы:
      a. **metadata** - здесь определён лейбл `app: mysql`, который будет применён ко всем подам, созданным этим StatefulSet.
      b. **spec** - описание контейнера, который будет развернут в поде. В данном случае это контейнер с mysql:
         i. **name** - имя контейнера - `mysql`.
         ii. **image** - имя образа, который будет использоваться для контейнера. В данном случае это официальный образ mysql версии `8.4.3` (взять из списка версий на docker hub, как “ну последний 8 версии, точно стабильный”).

iii. **env** - определяет переменные окружения для контейнера. Здесь указана переменная окружения `MYSQL_ROOT_PASSWORD`, которая задаёт пароль для пользователя `root` в mysql.

iv. **ports** - описание портов, которые контейнер будет слушать. В данном случае это порт `3306`, стандартный порт для mysql.

v. **volumeMounts**: Определяет, какие тома будут смонтированы в контейнер. Здесь указывается, что том с именем `mysql-data` будет монтироваться в путь `/var/lib/mysql` внутри контейнера. Это стандартное место для хранения данных mysql.

Применили манифест `kubectl apply -f mysql-statefulset.yaml`

mysql-service

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service # должно совпадать с serviceName в Stat
efulSet
  namespace: dev
spec:
  ports:
  - port: 3306
    targetPort: 3306
  clusterIP: None
  selector:
    app: mysql
```

Шпаргалка для себя (можно не читать):

1. **apiVersion: v1**

   Это версия API, используемая для описания объектов типа **Service** в Kubernetes. В данном случае используется версия `v1`, которая поддерживает базовые объекты, такие как **Pod**, **Service**, **ConfigMap** и тд.

2. **kind: Service**
   Service
   - это объект, который предоставляет стабильный доступ к наборам подов в Kubernetes. Сервис действует как абстракция для подов, делая их доступными для других компонентов, например, через DNS-имя или IP-адрес. Он может работать на основе различных типов (ClusterIP, NodePort, LoadBalancer), в зависимости от нужд.
3. **metadata**

   В разделе `metadata` задаются метаданные для ресурса:

   a. **name** - имя сервиса. В данном случае это `mysql-service`. Это имя будет использоваться для обращения к сервису внутри кластера kubernetes.

   b. **namespace** - hространство имён, в котором сервис будет создан. В этом случае это пространство имён `dev`, что **соответствует пространству, где развернут и StatefulSet с mysql**.
4. **spec**

   Раздел `spec` описывает характеристики самого сервиса:

   a. **ports**: Это описание портов, через которые сервис будет доступен:

      i. **port** - это внешний порт сервиса, который будет доступен для других компонентов kubernetes. В данном случае это порт `3306`, который является стандартным портом для mysql.

      ii. **targetPort** - это порт внутри контейнера, к которому будет направлен трафик, получаемый на порту сервиса. В данном случае это также `3306`, так как контейнер mysql будет слушать именно на этом порту **(соответствует порту, который указан в StatefulSet)**.
5. **clusterIP: None**: Установка `clusterIP: None` означает, что сервис не будет иметь обычного IP-адреса, как это бывает с типом сервиса `ClusterIP`. Вместо этого сервис будет работать в режиме **Headless Service**. Это позволяет kubernetes не создавать балансировщика нагрузки, а напрямую связываться с подами, предоставляя каждому поду уникальный DNS-ресурс. Это важно для **StatefulSet**, так как каждый под должен иметь

свой уникальный адрес для хранения состояния, например, для базы данных.

6. **selector** - это поле указывает, какие поды будут обслуживаться этим сервисом. В данном случае используется лейбл `app: mysql`. Это означает, что сервис будет направлять трафик к тем подам, которые имеют лейбл `app: mysql`, а именно к подам, созданным StatefulSet с mysql **(тут крч главное не запутаться в лейблах и везде указать правильный)**.

Применили манифест `kubectl apply -f mysql-service.yaml`

wordpress

wordpress-deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-deploy
  namespace: dev
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
      - name: wordpress
        image: wordpress:latest
        env:
        - name: WORDPRESS_DB_HOST
          value: mysql-service # должно совпадать с serviceNa
me в StatefulSet и name в Service mysql
```

```
        - name: WORDPRESS_DB_PASSWORD
          value: rootpassword
        ports:
        - containerPort: 80
```

Тут проговорим только некоторые моменты, иначе это превратится в огромную методичку еще на 50 страниц.

Не указываем serviceName, потому что он нужен для создание DNS записей, где нам важны имена подов. Так как тут деплой, который “легкозаменяем”, то такое делать не нужно.

Необходимо указать обращение ко всем репликам mysql через имя сервиса, который их “объединяет”

Применяем манифест `kubectl apply -f wordpress-deployment.yaml`

wordpress-service

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  namespace: dev
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
  selector:
    app: wordpress
```

NodePort - тип сервиса, который позволяет доступ к подам из внешней сети. Этот сервис создает порт на всех узлах кластера и перенаправляет трафик с этого порта в сервис внутри кластера, который, в свою очередь, маршрутизирует трафик к подам. По этому порту мы может обращаться на ip кластера и попадать в нужный сервис.

Применили манифест `kubectl apply -f wordpress-service.yaml`:

Проверка работы всей лабы

```
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ ls
mysql-service.yaml  mysql-statefulset.yaml  wordpress-deployment.yaml  wordpress-service.yaml
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl apply -f mysql-statefulset.yaml
statefulset.apps/mysql-set created
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl apply -f mysql-service.yaml
service/mysql-service created
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl apply -f wordpress-deployment.yaml
deployment.apps/wordpress-deploy created
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl apply -f wordpress-service.yaml
service/wordpress created
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$
```

1. проверяем состояние подов `kubectl get pods -n dev`

   *Тут столкнулся в проблемой volume, что его надо создавать. На данном этапе удалил просто эту часть из манифеста, так как дальше по лабораторным надо проделать эту часть.

```
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl get pods -n dev
NAME                                READY   STATUS    RESTARTS   AGE
mysql-set-0                         1/1     Running   0          2m31s
wordpress-deploy-76f64fb9f-xt6m5    1/1     Running   0          19m
```

2. проверяем состояние сервисов `kubectl get services -n dev`

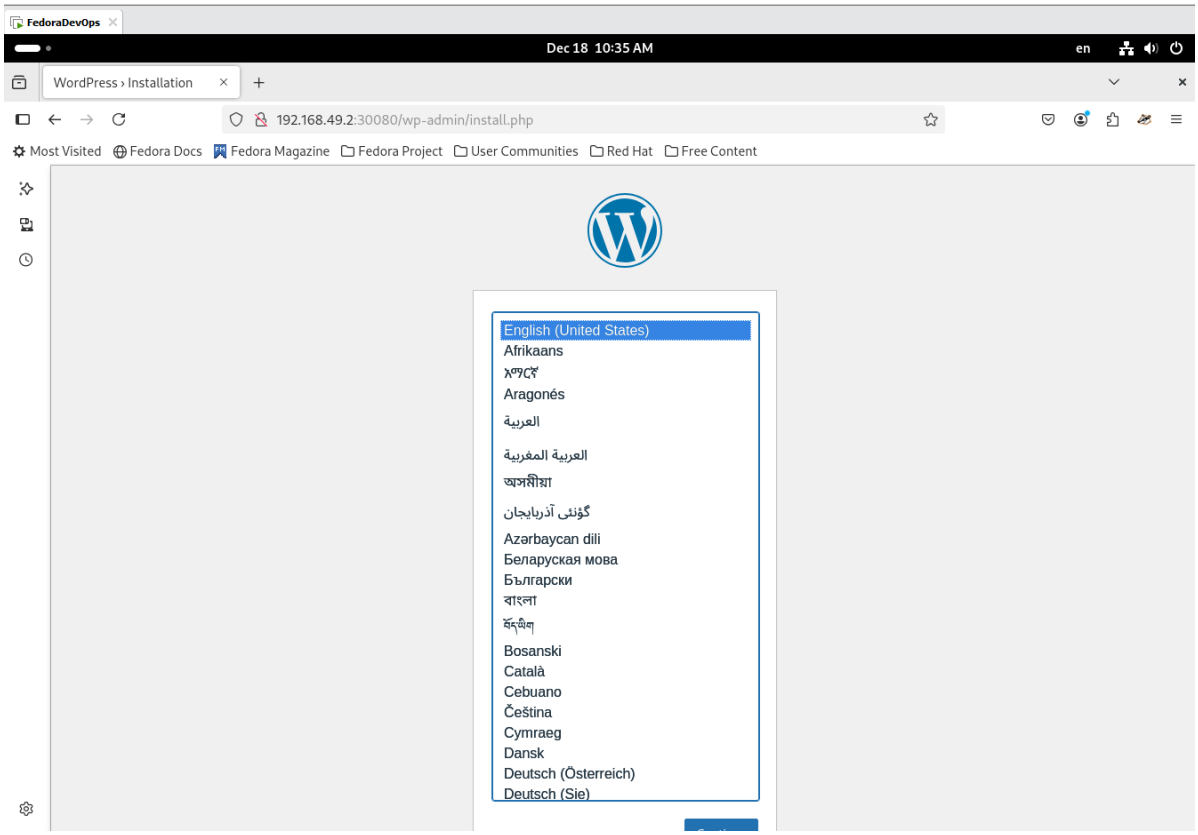
```
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ kubectl get services -n dev
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
mysql-service   ClusterIP   None           <none>        3306/TCP       3m37s
wordpress       NodePort    10.96.192.19   <none>        80:30080/TCP   3m16s
```

3. узнаем адрес узла кластера `minikube ip`

```
┌──(daniil@fedora-devops)-[~/DevOps/codeby_devops/lesson20][lesson20*]
└─$ minikube ip
192.168.49.2
```

4. проверяем доступность wordpress `192.168.49.2:30080`

   Ну и после такой поплавы, естественно, забыл прописать переменные для mysql и wp, в манифесте будут измененные.



Боль: честно говоря, очень много информации, чтобы ее быстро переварить. Про DNS записи, создаваемые кубером в headless режиме, чтобы мы могли обращаться к каждой реплике через имя сервиса, и что через это имя сервиса wp может обращаться ко всем этим репликам - прям заставило подразобраться посидеть. А тут еще по 2 лабы каждую неделю)