



МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Цифровая кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 11

по дисциплине

«Непрерывная разработка и интеграция CI/CD»

Тема практической работы: Администрирование кластера k3s

Выполнил студент группы 14

Стока И.П.

Руководитель практической работы

Волков М.Ю.

Практическая работа выполнена

«__»_____202__ г.

«Зачтено»

«__»_____202__ г.

Москва 2023г.

Кластеры: pods, replicaset, deployment

Создание кластера будет происходить в k3s, урезанной версии k8s. Для начал необходимо установить данное ПО и сконфигурировать для дальнейшей работы (Рисунок 1).

```
[johnstoka@fedora k3s]$ K3S_TOKEN=qwerty docker-compose up -d
k3s_server_1 is up-to-date
k3s_agent_1 is up-to-date
[johnstoka@fedora k3s]$ cp kubeconfig.yaml ~/k3s/kubeconfig.yaml
cp: 'kubeconfig.yaml' и '/home/johnstoka/k3s/kubeconfig.yaml' - один и тот же
файл
[johnstoka@fedora k3s]$ export KUBECONFIG=kubeconfig.yaml
[johnstoka@fedora k3s]$ kubectl get pods
No resources found in default namespace.
[johnstoka@fedora k3s]$
```

Рисунок 1 – Создание кластера

Наименьшими единицами в кластерах являются поды (pods), также существуют более усовершенствованные элементы как реплики (replicaset) и деплои (deployment). Ниже представлено их создание (Рисунок 2 - 4).

```
[johnstoka@fedora k3s]$ kubectl apply -f ~/k3s/pod.yaml
pod/my-pod unchanged
pod/my-pod-2 created
[johnstoka@fedora k3s]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-pod	1/1	Running	0	8m8s
my-pod-2	0/1	ContainerCreating	0	13s

Рисунок 2 – Создание подов

```
[johnstoka@fedora k3s]$ kubectl apply -f ~/k3s/replicaset.yaml
replicaset.apps/my-replicaset created
[johnstoka@fedora k3s]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-replicaset-pf7wq	0/1	ContainerCreating	0	14s
my-replicaset-d9nvn	1/1	Running	0	14s

```
[johnstoka@fedora k3s]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-replicaset-pf7wq	0/1	ContainerCreating	0	20s
my-replicaset-d9nvn	1/1	Running	0	20s

```
[johnstoka@fedora k3s]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-replicaset-d9nvn	1/1	Running	0	28s
my-replicaset-pf7wq	1/1	Running	0	28s

```
[johnstoka@fedora k3s]$ kubectl get replicaset
```

NAME	DESIRED	CURRENT	READY	AGE
my-replicaset	2	2	2	52s

```
[johnstoka@fedora k3s]$
```

Рисунок 3 – Создание реплики

```
[johnstoka@fedora k3s]$ kubectl apply -f deployment.yaml
deployment.apps/my-deployment unchanged
[johnstoka@fedora k3s]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment-7f59cd4667-tbhlh     1/1     Running   0           32s
my-deployment-7f59cd4667-bm79r     1/1     Running   0           32s
[johnstoka@fedora k3s]$ kubectl get replicaset
NAME                                DESIRED   CURRENT   READY   AGE
my-deployment-7f59cd4667           2         2         2       85s
[johnstoka@fedora k3s]$ kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
my-deployment   2/2     2             2           3m8s
[johnstoka@fedora k3s]$
```

Рисунок 4 – Создание деплоя

Далее приведен вход в контейнер (Рисунок 5).

```
[johnstoka@fedora k3s]$ kubectl exec -it my-deployment-7c996d865f-t2jg4 -- /bin/sh
# ls -l
total 4
drwxr-xr-x. 1 root root 706 May  9 2022 bin
drwxr-xr-x. 1 root root  0 Mar 19 2022 boot
drwxr-xr-x. 5 root root 360 Apr 20 18:04 dev
drwxr-xr-x. 1 root root  54 May 17 2022 docker-entrypoint.d
-rwxrwxr-x. 1 root root 1202 May 17 2022 docker-entrypoint.sh
drwxr-xr-x. 1 root root  20 Apr 20 18:04 etc
drwxr-xr-x. 1 root root  0 Mar 19 2022 home
drwxr-xr-x. 1 root root  46 May  9 2022 lib
drwxr-xr-x. 1 root root  40 May  9 2022 lib64
drwxr-xr-x. 1 root root  0 May  9 2022 media
drwxr-xr-x. 1 root root  0 May  9 2022 mnt
drwxr-xr-x. 1 root root  0 May  9 2022 opt
dr-xr-xr-x. 340 root root  0 Apr 20 18:04 proc
drwx----- 1 root root  30 May  9 2022 root
drwxr-xr-x. 1 root root  32 Apr 20 18:04 run
drwxr-xr-x. 1 root root 1008 May  9 2022 sbin
drwxr-xr-x. 1 root root  0 May  9 2022 srv
dr-xr-xr-x. 13 root root  0 Apr 20 16:51 sys
drwxrwxrwt. 1 root root  0 May 17 2022 tmp
drwxr-xr-x. 1 root root  40 May  9 2022 usr
drwxr-xr-x. 1 root root  10 May  9 2022 var
# exit
[johnstoka@fedora k3s]$
```

Рисунок 5 - Запуск контейнера

Также представлено содержимое элементов кластера (Рисунок 6 - 8).

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - image: nginx:1.20
    name: nginx
    ports:
    - containerPort: 80
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-2
spec:
  containers:
  - image: alpine:3.14
    name: alpine
    ports:
    - containerPort: 80
```

Рисунок 6 – Содержимое pod.yml

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - image: nginx:1.20
        name: nginx
        ports:
        - containerPort: 80

```

Рисунок 7 – Содержимое replicaset.yml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - image: nginx:1.20
        name: nginx
        ports:
        - containerPort: 80

```

Рисунок 8 – Содержимое deployment.yml

Также можно обновлять элементы деплоев или реплик, например, в приведенных примерах обновлены образы nginx с 1.20 до 1.21 (Рисунок 9 - 10).

```

[johnstoka@fedora k3s]$ kubectl set image deployment my-deployment nginx=nginx:1.21
deployment.apps/my-deployment image updated
[johnstoka@fedora k3s]$ kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
my-deployment       2/2     2            2           5m2s
[johnstoka@fedora k3s]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment-8479df6dc6-ctdbk     1/1     Running   0          69s
my-deployment-8479df6dc6-c2tx5     1/1     Running   0          62s
[johnstoka@fedora k3s]$ kubectl describe pod my-deployment-8479df6dc6-ctdbk | grep i
mage
    Normal Pulled      107s kubelet                                Container image "nginx:1.21" already p
resent on machine
[johnstoka@fedora k3s]$

```

Рисунок 9 – Обновление образа в deployment

```
[johnstoka@fedora k3s]$ kubectl get replicaset
NAME          DESIRED    CURRENT    READY    AGE
my-replicaset 2           2           2        7m4s
[johnstoka@fedora k3s]$ kubectl describe pod my-replicaset | grep image
Normal Pulled 7m36s kubelet Container image "nginx:1.20" already
present on machine
Normal Pulling 7m36s kubelet Pulling image "nginx:1.20"
Normal Pulled 7m16s kubelet Successfully pulled image "nginx:1.20"
" in 20.40678413s (20.406801436s including waiting)
[johnstoka@fedora k3s]$ kubectl delete pod --all
pod "my-replicaset-d9nvn" deleted
pod "my-replicaset-pf7wq" deleted
[johnstoka@fedora k3s]$ kubectl get pods
NAME          READY    STATUS             RESTARTS    AGE
my-replicaset-wnrm9 0/1      ContainerCreating   0            17s
my-replicaset-jn5t6 0/1      ContainerCreating   0            16s
[johnstoka@fedora k3s]$ kubectl get pods
NAME          READY    STATUS    RESTARTS    AGE
my-replicaset-wnrm9 1/1      Running   0            35s
my-replicaset-jn5t6 1/1      Running   0            34s
[johnstoka@fedora k3s]$ kubectl describe pod my-replicaset | grep image
Normal Pulling 37s kubelet Pulling image "nginx:1.21"
Normal Pulled 7s kubelet Successfully pulled image "nginx:1.21"
in 29.491988908s (29.492010274s including waiting)
Normal Pulling 36s kubelet Pulling image "nginx:1.21"
Normal Pulled 7s kubelet Successfully pulled image "nginx:1.21"
in 29.211720886s (29.211779996s including waiting)
[johnstoka@fedora k3s]$
```

Рисунок 10 – Обновление образа в replicaset

Существует возможность установления ограничений по ресурсам (Рисунок 11).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - image: nginx:1.20
          name: nginx
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: 0.5
              memory: 100Mi
            limits:
              cpu: 0.5
              memory: 100Mi
```

Рисунок 11 – Добавление ограничений в deployment.yml

Далее проверяется внесение изменений (Рисунок 12).

```
[johnstoka@fedora k3s]$ kubectl apply -f deployment.yaml
deployment.apps/my-deployment configured
[johnstoka@fedora k3s]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment-7c996d865f-t2jg4      1/1     Running   0           21s
my-deployment-7c996d865f-z4thz      1/1     Running   0           19s
[johnstoka@fedora k3s]$ kubectl describe pod my-deployment-7c996d865f-t2jg4 | grep c
pu
      cpu:          500m
      cpu:          500m
[johnstoka@fedora k3s]$ kubectl describe pod my-deployment-7c996d865f-t2jg4 | grep m
emory
      memory:       100Mi
      memory:       100Mi
[johnstoka@fedora k3s]$
```

Рисунок 13 – Проверка внесения ограничений

Для проверки работоспособности сервиса используется следующая команда (Рисунок 14).

```
[johnstoka@fedora k3s]$ kubectl port-forward my-deployment-7c996d865f-t2jg4 8000:80
Forwarding from 127.0.0.1:8000 -> 80
Forwarding from [::1]:8000 -> 80
Handling connection for 8000
```

Рисунок 14 – Проверка работы сервиса

Проверим соединение по проложенному порту (Рисунок 15).



Рисунок 15 – Проверка localhost:8000

И наконец представлена реализация удаления элементов кластера (Рисунок 16).

```
[johnstoka@fedora k3s]$ kubectl delete pod --all  
pod "my-pod" deleted  
pod "my-pod-2" deleted
```

Рисунок 16 – Удаление подов

```
[johnstoka@fedora k3s]$ kubectl delete replicaset --all  
replicaset.apps "my-replicaset" deleted  
replicaset.apps "my-deployment-7f59cd4667" deleted  
[johnstoka@fedora k3s]$
```

Рисунок 17 – Удаление реплик

```
[johnstoka@fedora k3s]$ kubectl delete deployment --all  
deployment.apps "my-deployment" deleted
```

Рисунок 18 – Удаление деплоев

Deployment: secret, env, configmap

При повторном использовании k3s, необходимо загрузить kubeconfig (Рисунок 19).

```
[johnstoka@fedora k3s]$ export KUBECONFIG=~/.k3s/kubeconfig.yaml  
[johnstoka@fedora k3s]$ kubectl get pods  
No resources found in default namespace.
```

Рисунок 19 – Экспорт конфига

В deployment поддерживается создание сред, с необходимыми полями, хранящих необходимые данные, тут configMapRef является ссылкой на использованный configmap (Рисунок 20).

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: my-deployment  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: my-app  
  strategy:  
    rollingUpdate:  
      maxSurge: 1  
      maxUnavailable: 1  
    type: RollingUpdate  
  template:  
    metadata:  
      labels:  
        app: my-app  
    spec:  
      containers:  
        - image: nginx:1.20  
          name: nginx  
          env:  
            - name: TEST  
              value: foo  
          envFrom:  
            - configMapRef:  
                name: my-configmap-env  
          ports:  
            - containerPort: 80  
          resources:  
            requests:  
              cpu: 50m
```

Рисунок 20 – Добавление среды

ConfigMap позволяет отделить артефакты конфигурации от содержимого образа (Рисунок 21).


```

apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap-env
data:
  dbhost: postgresql
  DEBUG: "false"

```

Рисунок 21 – Содержимое ConfigMap

```

[johnstoka@fedora k3s]$ kubectl apply -f configmap.yml
configmap/my-configmap-env created
[johnstoka@fedora k3s]$ kubectl apply -f deployment-with-env.yml
deployment.apps/my-deployment unchanged
[johnstoka@fedora k3s]$ kubectl get pods

```

NAME	READY	STATUS	RESTARTS	AGE
my-deployment-7d4bfc4d8f-vpjc	1/1	Running	0	82s

Рисунок 22 – Запуск deployment с configmap

Для использования приватных данных используются secret, которые шифруют данные даже при процессе CI/CD, пользователь может получить данные, но они будут закодированы. В данном примере используется secret с полями ключ и имя (Рисунок 23).

```

GNU nano 6.4 deployment-with-env.yml Изменён
replicas: 1
selector:
  matchLabels:
    app: my-app
strategy:
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
  type: RollingUpdate
template:
  metadata:
    labels:
      app: my-app
  spec:
    containers:
      - image: nginx:1.20
        name: nginx
        env:
          - name: TEST
            value: foo
          - name: env
            valueFrom:
              secretKeyRef:
                name: test
                key: test1
        ports:
          - containerPort: 80
    resources:
      requests:
        cpu: 50m
        memory: 100Mi
      limits:
        cpu: 100m

```

Рисунок 23 – Добавление модуля secret

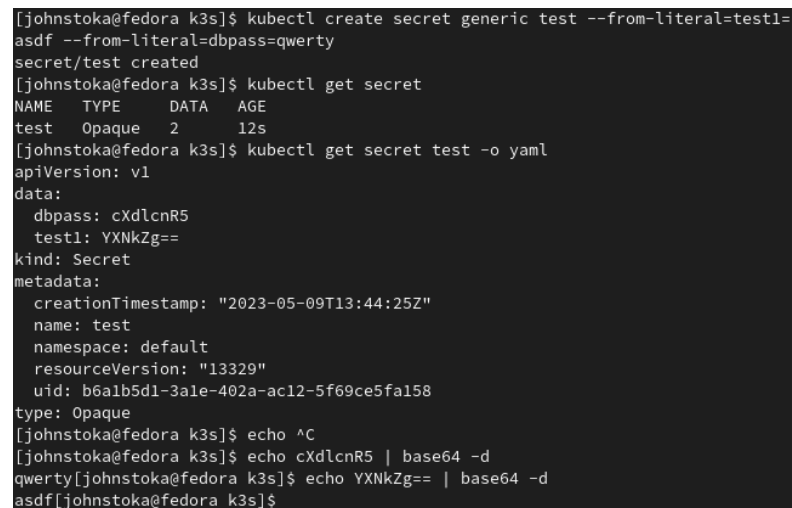
Далее представлено содержимое файла secret.yml (Рисунок 24).



```
johnstoka@fedora:~/k3s — sudo nano secret.yml
GNU nano 6.4 secret.yml
apiVersion: v1
kind: Secret
metadata:
  name: test
stringData:
  test1: updated
```

Рисунок 24 – Содержимое secret

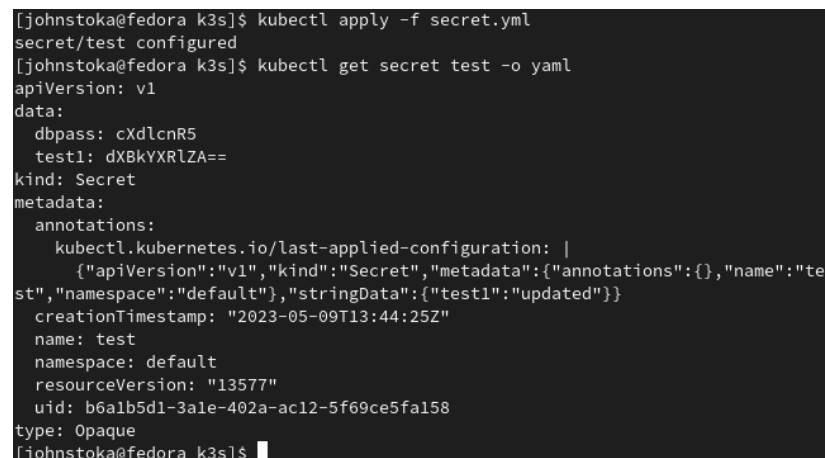
Реализация создания секрета представлено далее (Рисунок 25).



```
[johnstoka@fedora k3s]$ kubectl create secret generic test --from-literal=test1=
asdf --from-literal=dbpass=qwerty
secret/test created
[johnstoka@fedora k3s]$ kubectl get secret
NAME      TYPE      DATA      AGE
test      Opaque    2          12s
[johnstoka@fedora k3s]$ kubectl get secret test -o yaml
apiVersion: v1
data:
  dbpass: cXdlnR5
  test1: YXNkZg==
kind: Secret
metadata:
  creationTimestamp: "2023-05-09T13:44:25Z"
  name: test
  namespace: default
  resourceVersion: "13329"
  uid: b6a1b5d1-3a1e-402a-ac12-5f69ce5fa158
type: Opaque
[johnstoka@fedora k3s]$ echo ^C
[johnstoka@fedora k3s]$ echo cXdlnR5 | base64 -d
qwerty[johnstoka@fedora k3s]$ echo YXNkZg== | base64 -d
asdf[johnstoka@fedora k3s]$
```

Рисунок 25 – Создание секрета

Далее проводится запуск deployment с модулем secret (Рисунок 26).



```
[johnstoka@fedora k3s]$ kubectl apply -f secret.yml
secret/test configured
[johnstoka@fedora k3s]$ kubectl get secret test -o yaml
apiVersion: v1
data:
  dbpass: cXdlnR5
  test1: dXBkYXRlZA==
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Secret","metadata":{"annotations":{},"name":"te
st","namespace":"default"},"stringData":{"test1":"updated"}}
  creationTimestamp: "2023-05-09T13:44:25Z"
  name: test
  namespace: default
  resourceVersion: "13577"
  uid: b6a1b5d1-3a1e-402a-ac12-5f69ce5fa158
type: Opaque
[johnstoka@fedora k3s]$
```

Рисунок 26 – Запуск secret

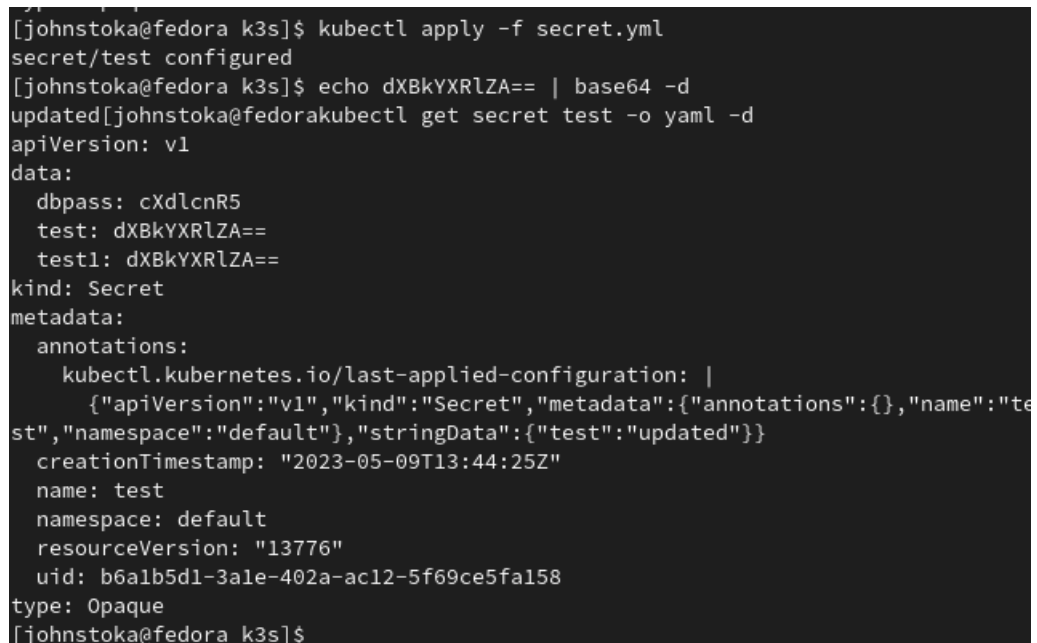
Для изменения ключа или его добавления достаточно изменить secret файл, а именно поля с metadata, добавляя их или же изменяя (было test1, стало test) (Рисунок 27).



```
johnstoka@fedora:~/k3s — sudo nano secret.yml
GNU nano 6.4 secret.yml
apiVersion: v1
kind: Secret
metadata:
  name: test
stringData:
  test: updated
```

Рисунок 27 – Добавления ключа

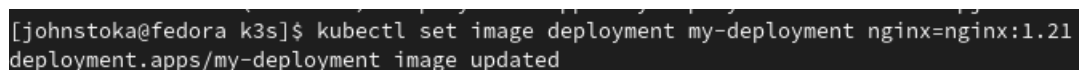
Далее проверяется внесение изменений, как видно на рисунке, произошло добавление test (Рисунок 28).



```
[johnstoka@fedora k3s]$ kubectl apply -f secret.yml
secret/test configured
[johnstoka@fedora k3s]$ echo dXBkYXRlZA== | base64 -d
updated[johnstoka@fedorakubectl get secret test -o yaml -d
apiVersion: v1
data:
  dbpass: cXdlcnR5
  test: dXBkYXRlZA==
  test1: dXBkYXRlZA==
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Secret","metadata":{"annotations":{},"name":"test","namespace":"default"},"stringData":{"test":"updated"}}
  creationTimestamp: "2023-05-09T13:44:25Z"
  name: test
  namespace: default
  resourceVersion: "13776"
  uid: b6a1b5d1-3a1e-402a-ac12-5f69ce5fa158
type: Opaque
[johnstoka@fedora k3s]$
```

Рисунок 28 – Проверка внесения изменений

Для обновления образа nginx используется следующая команда (Рисунок 29).



```
[johnstoka@fedora k3s]$ kubectl set image deployment my-deployment nginx=nginx:1.21
deployment.apps/my-deployment image updated
```

Рисунок 29 – Обновление версии образа nginx

Deployment: hostpath, emptydir, pv, pvc, probes

Для хранения данных кластера используются разные технологии. Volumes является более масштабированным и удобным выбором, но также существуют механизмы hostpath и emptydir, метод которых заключается в хранении данных на одном узле, если же кластер переходит на другой узел, то данные теряются, поэтому данные механизмы хранения используются на короткий промежуток времени, emptydir позволяет очистить память, используемую кластером, поэтому данный механизм удобен при работе в течении сессии, после которой данные не предоставляют большой ценности.

Далее представлен запуск deployment с использованием разным механизмов хранения (Рисунок 30), сами же конфигурационные файлы представлены на github.

```
[johnstoka@fedora k3s]$ kubectl apply -f deployment_hostPath.yml
deployment.apps/my-deployment created
[johnstoka@fedora k3s]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment-688c4bbb4b-p4lph      1/1     Running   0           44s
[johnstoka@fedora k3s]$ kubectl exec -it my-deployment-688c4bbb4b-p4lph -- df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          50G   8.1G   41G   17% /
tmpfs            64M    0    64M    0% /dev
overlay          50G   8.1G   41G   17% /files
/dev/sda3        50G   8.1G   41G   17% /etc/hosts
shm              64M    0    64M    0% /dev/shm
tmpfs           100M   12K   100M    1% /run/secrets/kubernetes.io/serviceaccount
tmpfs           985M    0   985M    0% /proc/acpi
tmpfs           985M    0   985M    0% /proc/scsi
tmpfs           985M    0   985M    0% /sys/firmware
[johnstoka@fedora k3s]$
```

Рисунок 30 – Запуск deployment с hostpath

Также приводится пример добавления записей в emptydir (Рисунок 31).

```
[johnstoka@fedora k3s]$ kubectl apply -f deployment_emptyDir.yml
deployment.apps/my-deployment created
[johnstoka@fedora k3s]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment-8cfbc5ff7-rv8pw      1/1     Running   0           9s
[johnstoka@fedora k3s]$ kubectl exec -it my-deployment-8cfbc5ff7-rv8pw -- sh -c
'echo hello > /files/data.txt'
[johnstoka@fedora k3s]$ kubectl exec -it my-deployment-8cfbc5ff7-rv8pw -- cat /f
iles/data.txt
hello
[johnstoka@fedora k3s]$
```

Рисунок 31 – Запись в deployment с emptydir

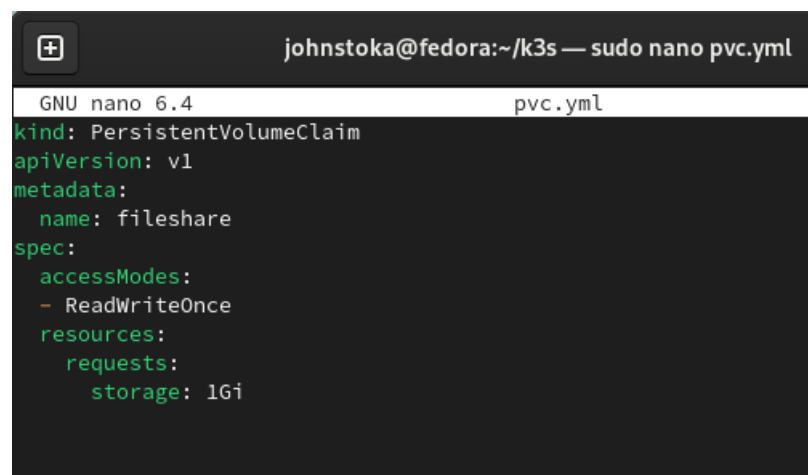
PersistentVolume – это часть хранилища в кластере, которая была подготовлена администратором или динамически подготавливается с помощью классов хранилища.

PersistentVolumeClaim – это запрос пользователя на хранение. Модули потребляют ресурсы узлов, а pvc ресурсы pv.

Далее представлено добавление модуля pvc в deployment, а также его содержимое (Рисунок 32 - 33).

```
type: RollingUpdate
template:
  metadata:
    labels:
      app: my-app
  spec:
    containers:
      - image: nginx:1.20
        name: nginx
        env:
          - name: TEST
            value: foo
          - name: env
            valueFrom:
              secretKeyRef:
                name: test
                key: test1
        ports:
          - containerPort: 80
        resources:
          requests:
            cpu: 50m
            memory: 100Mi
          limits:
            cpu: 100m
            memory: 100Mi
        volumeMounts:
          - name: data
            mountPath: /files
    volumes:
      - name: data
        persistentVolumeClaim:
          claimName: fileshare
```

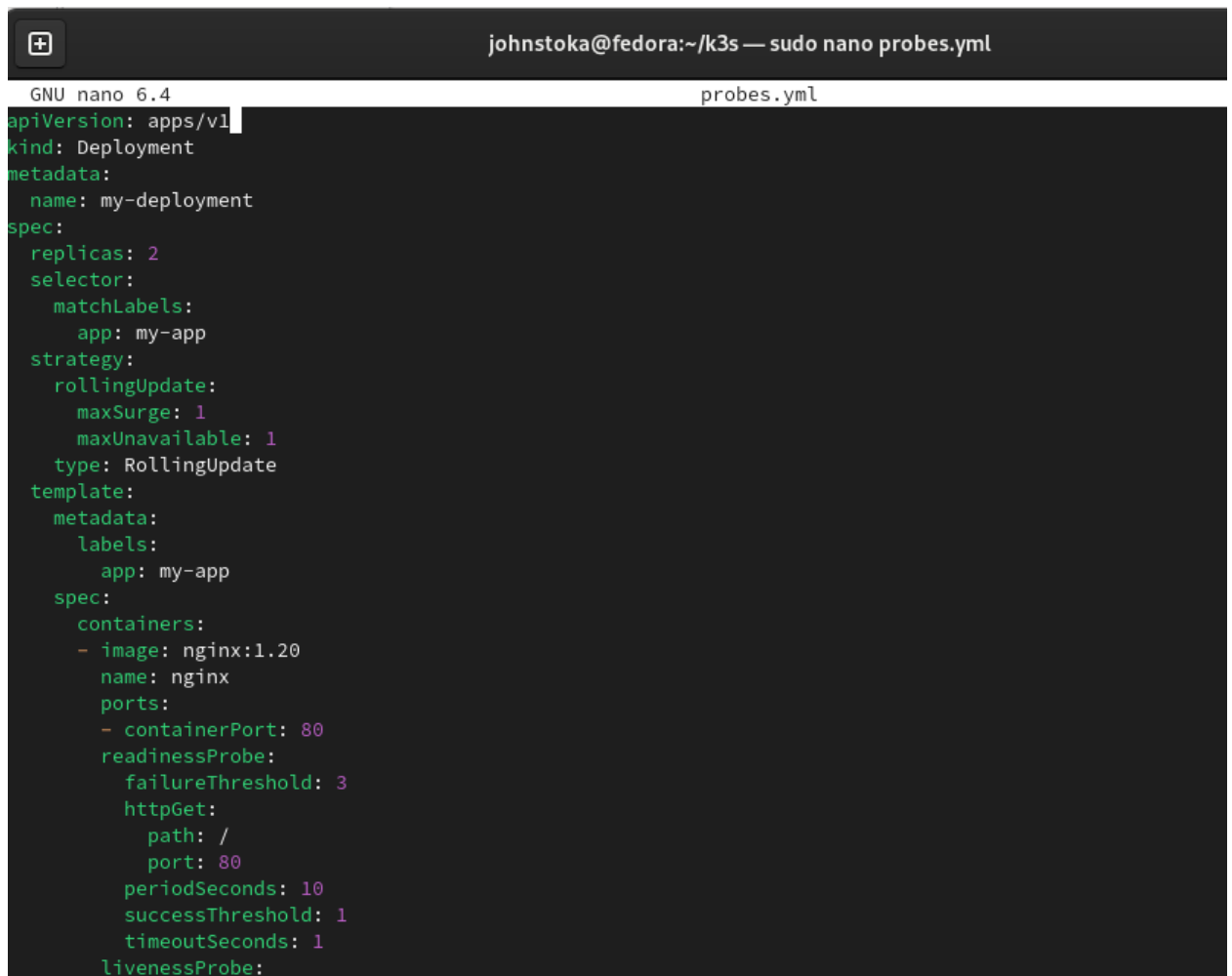
Рисунок 32 – Добавление модуля pvc



```
johnstoka@fedora:~/k3s — sudo nano pvc.yml
GNU nano 6.4 pvc.yml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: fileshare
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Рисунок 32 – Содержимое pvc

Probes необходим для отлавливания ошибок, некий датчик живучести и работоспособности, например, при нахождении в кластере бд и веб-сервера, бд начинает работу с задержкой, чтобы веб-сервер не отправлял запросы которые не будут задействованы в бд используется данный инструмент. В данном примере probes состоит из 3 частей: readiness probe, liveness probe и startup probe. Period seconds это как часто (в секундах) выполнять зонд (probe), successThreshold указывает сколько успешных проверок необходимо, failureThreshold означает количество неудачных проверок, после которых кластер будет перезагружен, timeoutSeconds это количество секунд, по истечении которого истекает время ожидания зонда (probe), InitialDelaySeconds это время задержки перед запуском кластера.



```
johnstoka@fedora:~/k3s — sudo nano probes.yml
GNU nano 6.4 probes.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - image: nginx:1.20
        name: nginx
        ports:
        - containerPort: 80
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /
            port: 80
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        livenessProbe:
```

Рисунок 33 – Содержимое probes

Далее производится запуск probes (Рисунок 34).

```
[johnstoka@fedora k3s]$ kubectl apply -f probes.yml
deployment.apps/my-deployment configured
[johnstoka@fedora k3s]$ kubectl get probes.yml
error: the server doesn't have a resource type "probes"
[johnstoka@fedora k3s]$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
my-deployment-84b59675b-cb6nr	1/1	Running	0	59s
my-deployment-84b59675b-hjpv1	1/1	Running	0	58s

Рисунок 34 – Запуск probes