

# K8s Practice

---

## ШАГ 1. Устанавливаем Docker, kubectl и kind

### 1.1 Установка Docker

(на Ubuntu / Debian)

```
sudo apt update
sudo apt install -y docker.io
# запускаем и включаем автозапуск
sudo systemctl enable --now docker
# проверяем
docker --version
sudo docker run --rm hello-world # проверка запуска контейнера
```

Если `hello-world` успешно запустился — Docker OK.

Если возникает `permission denied` — можно добавить пользователя в группу docker:

```
sudo usermod -aG docker $USER
# затем выйдите/войдите снова или выполните:
newgrp docker
```

---

### 1.2 Установка kubectl (быстрый и надёжный способ)

```
curl -LO "https://dl.k8s.io/release/v1.30.0/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client --short
```

версия v1.30.0 в примере — можно заменить на актуальную, но для урока эта стабильная.

Если нет прав в `/usr/local/bin`, используйте `/usr/bin` или установите через пакетный менеджер.

## 1.3 Установка kind (рекомендую бинарную установку — не требуется Go)

```
curl -Lo ./kind "https://kind.sigs.k8s.io/dl/v0.22.0/kind-linux-amd64"
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
kind --version
```

(Если предпочитаете `go install`, это тоже работает, но требует установленного Go — поэтому бинарный метод проще на занятии.)

## 1.4 Проверки

```
docker ps -a      # docker работает
kubectl version --client --short
kind --version
```

Если кто-то видит ошибки — разберём их отдельно (Docker не запущен; права; скачивание блокируется корпоративным прокси и т.п.).

# ШАГ 2. Создаём кластер Kind

## 2.1 Создать простой кластер

```
kind create cluster --name demo
```

**Ожидаемое поведение:** kind создаст контейнеры Docker (они будут имитировать Kubernetes-ноды). Команда вернёт несколько строк про

создание control-plane и node, а также напомним, как проверить контекст kubectl.

## 2.2 Проверка нод и контекста

```
kubectl cluster-info  
kubectl get nodes  
kubectl config current-context
```

**Ожидаемый вывод:** `kubectl get nodes` покажет одну ноду `demo-control-plane` в статусе `Ready`.

Если нода `NotReady` — подождать минуту и проверить `kubectl get pods -A` (все Pod'ы в kube-system могут стартовать). Если долго `CrashLoopBackOff` — смотреть логи и Docker контейнеры `docker ps`.

## ШАГ 3. Запускаем первый Pod

Важно: в современных версиях `kubectl run` по умолчанию создаёт **Deployment** (restart policy = Always). Если мы хотим именно Pod, нужно указать `--restart=Never`.

### Вариант А — быстро: запустить Pod (одноразовый Pod)

```
kubectl run nginx-pod --image=nginx --port=80 --restart=Never  
kubectl get pods
```

**Ожидаемый вывод:** Pod `nginx-pod` в статусе `Running`.

### Вариант В — рекомендовано: создать Deployment (правильнее для продакшн/школы)

```
kubectl create deployment nginx-deploy --image=nginx  
kubectl get deployment  
kubectl get pods
```

Deployment создаст ReplicaSet и один Pod. Это лучший путь для масштабирования и обновлений.

**Deployment** — объект «высокого уровня», ReplicaSet поддерживает количество Pod'ов, Pod — эфемерная единица.

---

## ШАГ 4. Проверка Pod и базовая диагностика

### 4.1 Посмотреть состояние Pod

```
kubectl get pods
kubectl describe pod <pod-name>
```

В `describe` ищем Events: image pull error, insufficient memory, CrashLoopBackOff.

### 4.2 Просмотр логов

```
kubectl logs <pod-name>           # контейнер по-умолчанию (если один)
kubectl logs -f <pod-name>         # следим за логами в реальном времени
и
kubectl logs <pod-name> -c <container-name> # если в Pod несколько контейнеров
```

### 4.3 Выполнить команду внутри Pod

```
kubectl exec -it <pod-name> -- /bin/sh
# затем внутри:
ps aux
curl -I http://localhost:80
exit
```

---

## ШАГ 5. Доступ к nginx — порт-форвардинг и Service

### 5.1 Простой метод — порт-форвардинг

```
kubectl port-forward pod/<pod-name> 8080:80
# в другой терминале
curl http://localhost:8080
# или открыть в браузере http://localhost:8080
```

Если видим HTML страницы nginx — успешно.

**Примечание:** для Deployment нужно порт-форвардить Pod (узнать имя `kubectl get pods` ) или используем Service.

## 5.2 Альтернатива: создать Service типа ClusterIP (внутри кластера) и NodePort (для доступа извне)

ClusterIP (внутри кластера):

```
kubectl expose deployment nginx-deploy --port=80 --target-port=80 --name=nginx-svc --type=ClusterIP
kubectl get svc
```

NodePort (откроет порт на ноде; в kind он обычно доступен на localhost):

```
kubectl expose deployment nginx-deploy --type=NodePort --name=nginx-nodeport --port=80
kubectl get svc nginx-nodeport -o yaml
# смотрим allocated nodePort, например 30007
curl http://localhost:30007
```

(На kind NodePort часто проброшен на localhost — работает. Если нет, используем port-forward.)

## ШАГ 6. Пример YAML-манифеста Pod и разбор полей

`pod.yaml` (тот же, что в слайде):

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

- `apiVersion` — версия API (v1 для базовых объектов).
- `kind` — тип ресурса (Pod).
- `metadata` — имя, метки (labels) и аннотации.
- `spec` — спецификация: контейнеры, тома, политики перезапуска, probes и т.п.

как применить:

```
kubectl apply -f pod.yaml
kubectl get pods
```

## ШАГ 7. Полезные kubectl команды

<code>kubectl get pods</code>	# список Pod'ов
<code>kubectl get pods -o wide</code>	# более подробно (IP, Node)
<code>kubectl describe pod &lt;name&gt;</code>	# детальная диагностика и Events
<code>kubectl logs &lt;pod&gt;</code>	# посмотреть логи
<code>kubectl exec -it &lt;pod&gt; -- /bin/sh</code>	# зайти внутрь контейнера
<code>kubectl delete pod &lt;name&gt;</code>	# удалить Pod

```
kubectl apply -f file.yaml          # создать/обновить объекты из манифес
та
kubectl get svc                    # список сервисов
kubectl get deployment             # deployments и статус rollout
kubectl rollout status deployment/<name> # статус обновления
kubectl get events --sort-by='.metadata.creationTimestamp' # последние со
бытия
kubectl get all -n <namespace>    # все объекты в namespace
kubectl explain pod.spec           # подсказки по структуре
```

## ШАГ 8. Отработка ошибок и диагностика — самые частые проблемы

Говори студентам что проверять и как фиксит.

### Проблема: Pod в статусе **ImagePullBackOff**

- `kubectl describe pod <name>` → ищем event с `ErrImagePull` или `ImagePullBackOff`.
- Причины: опечатка в имени образа, приватный реестр (нужен `imagePullSecrets`), отсутствие интернета.
- Решение: поправить `image`, создать secret и прописать в Pod.

### Проблема: Pod в **CrashLoopBackOff**

- `kubectl logs <pod>` — смотрим, что падает.
- `kubectl describe pod` — смотрим events.
- Возможно, приложение завершает работу (нужна командная строка/args), или отсутствует `required env variables/volumes`.

### Проблема: **Node NotReady**

- `kubectl get nodes`
- Проверить systemd статус `docker/containerd` на хосте:

```
sudo systemctl status docker
```

- Посмотреть лог kubelet: `docker logs <kind-node-container>` или `journalctl -u kubelet` (если реальная нода).
- 

## ШАГ 9. Очистка после урока

Чтобы удалить всё и вернуть систему в исходное состояние:

```
kind delete cluster --name demo  
kubectl config get-contexts # убедиться, что контекст удалён
```

Удаление кластера освобождает Docker ресурсы.

---