

# Компоненты и архитектура кластера Kubernetes

---

Султанов Артур (Р3213)

# Что за зверь?

Kubernetes (K8s) — это открытое программное обеспечение для автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями.



# На практике

---

# Приложение

```
@app.route('/')
def hello():
    return 'I\'m Romashka!'

@app.route('/divisors')
def divisors():
    n = request.args.get('n')
    if n is None:
        return 'please provide n'

    try:
        n = int(n)
    except:
        return 'please provide an integer'

    divisors = find_divisors(n)

    return ', '.join(list(map(lambda n: str(n), divisors)))
```

```
> flask --app=app.server run -p 8000 &
> curl "localhost:8000"
I'm Romashka!
> curl "localhost:8000/divisors?n=123"
1, 3, 41, 123
```

# Обернем в docker

```
FROM python:3.11

RUN mkdir /www
ADD . /www

RUN groupadd --gid 3000 dolphin
RUN useradd --uid 3000 --gid dolphin --shell /bin/bash --create-home --home-dir /home/dolphin dolphin
RUN chown -R dolphin:dolphin /www

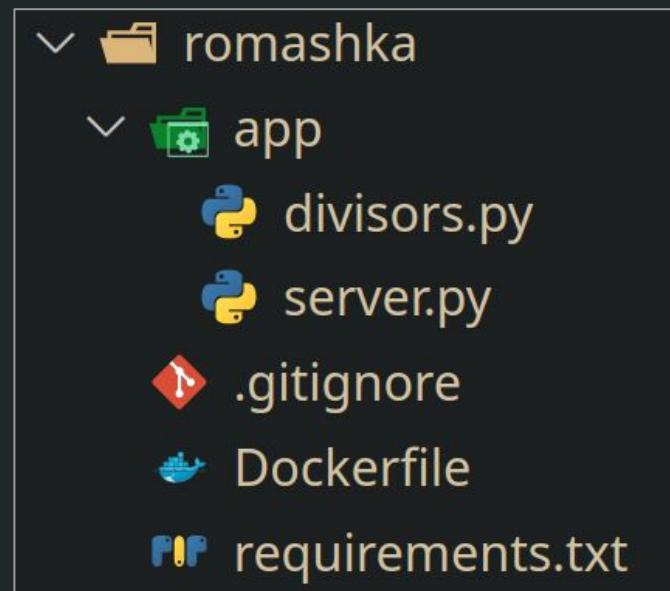
USER dolphin

WORKDIR /www

RUN python -m pip install -r requirements.txt

CMD python -m flask --app app.server run -h 0.0.0.0 -p 8000 --no-debugger
```

# Структура проекта



# Запустим контейнер

```
› docker buildx build --file Dockerfile --tag=romashka:1.0 --load . >/dev/null
› docker run -p 80:8000 -d --name=romashka --restart=always romashka:1.0 >/dev/null
c9f29a6aeeb0e312b398b262875f675731a8e95e080b169de003565c40b0607e
› curl "localhost:80"
I'm Romashka!
› curl "localhost:80/divisors?n=123"
1, 3, 41, 123
```

# Результат

123.123.123.123



Ура, прод готов. По домам?

---

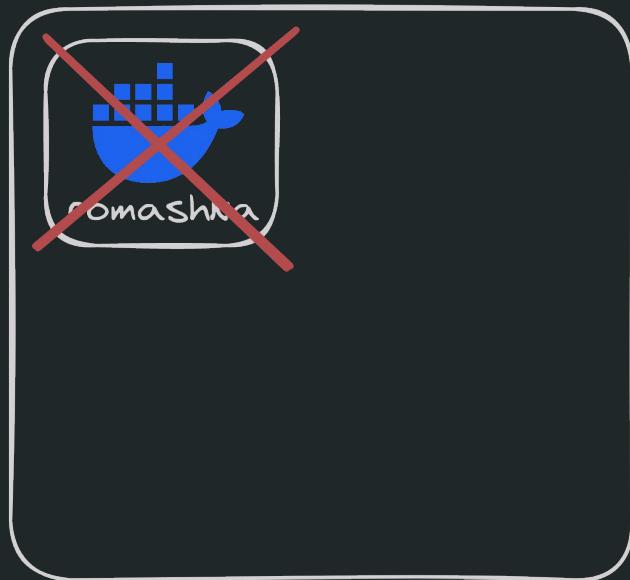
Ой! 1: Контейнер не резиновый

123.123.123.123



Ой! 2: Контейнер умер :с

123.123.123.123



# Ой! 3: Сервер временно недоступен

~~123.123.123.123~~



# Ой! 4: Сервер умер



# Ой! Это только начало

123.123.123.123  
*(unavailable 3 times a day)*



# Заворачиваем в “кубер” \*

---

\*: с учетом того, что кластер полностью настроен и готов к использованию

# Most sane YAML dev

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: romashka
spec:
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 2
  selector:
    matchLabels:
      app: romashka
  template:
    metadata:
      labels:
        app: romashka
    spec:
      containers:
        - name: romashka
          image: romashka:1.0
          imagePullPolicy: Never
          resources:
            limits:
              cpu: "100m"
              memory: "128Mi"
            requests:
              cpu: "50m"
              memory: "128Mi"
        ports:
          - name: http
            containerPort: 8000
        livenessProbe:
          httpGet:
            path: /
            port: 8000
        readinessProbe:
          httpGet:
            path: /
            port: 8000
```

```
apiVersion: v1
kind: Service
metadata:
  name: romashka
spec:
  type: ClusterIP
  selector:
    app: romashka
  ports:
    - name: http
      port: 8000
      targetPort: http
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: romashka
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: romashka
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: romashka
                port:
                  name: http
```

✓  infra

 deployment.yaml

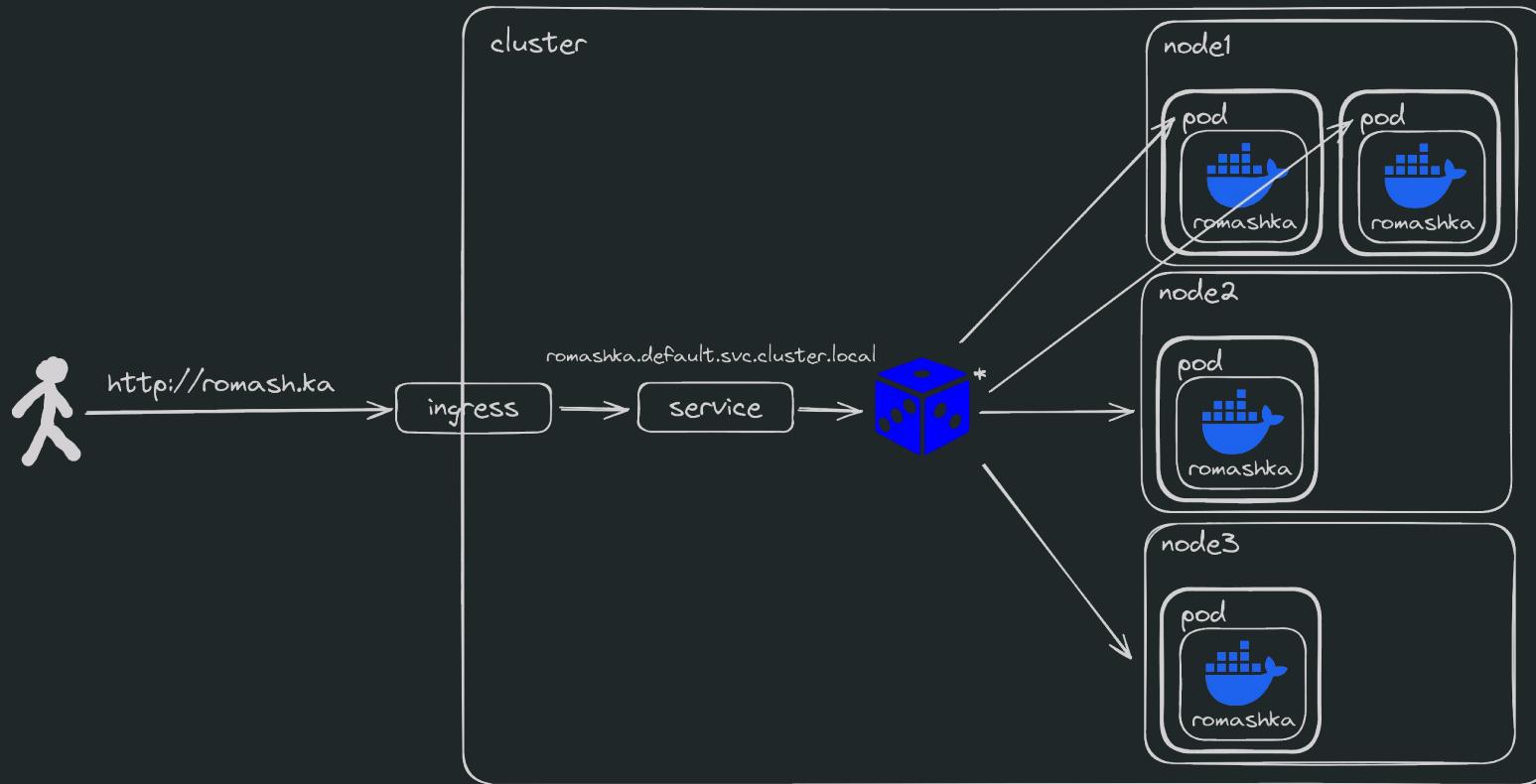
 ingress.yaml

 service.yaml

# Применим

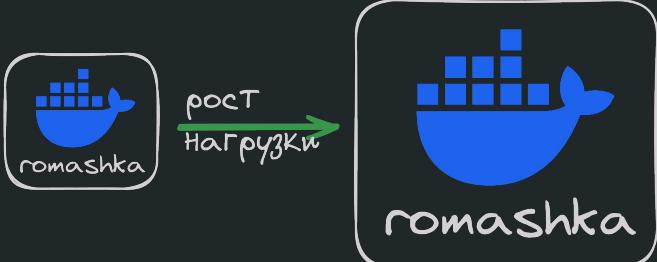
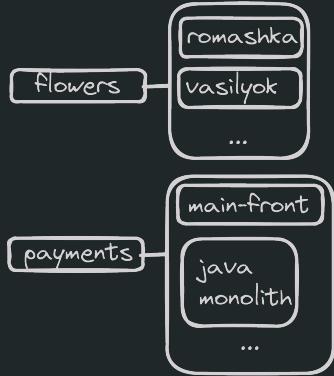
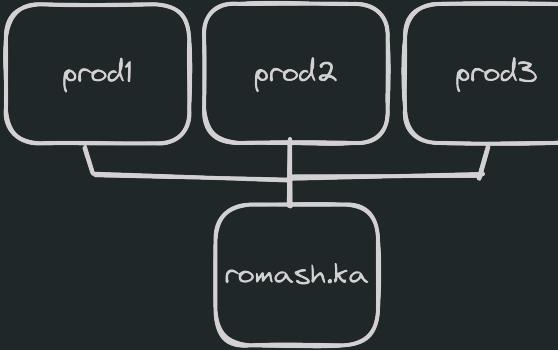
```
› kubectl apply -f infra/
› kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
romashka  4/4     4           4           1m
› curl "romash.ka:80"
I'm Romashka!
› curl "romash.ka:80/divisors?n=123"
1, 3, 41, 123
```

# Результат с точки зрения сущностей (не архитектуры!)





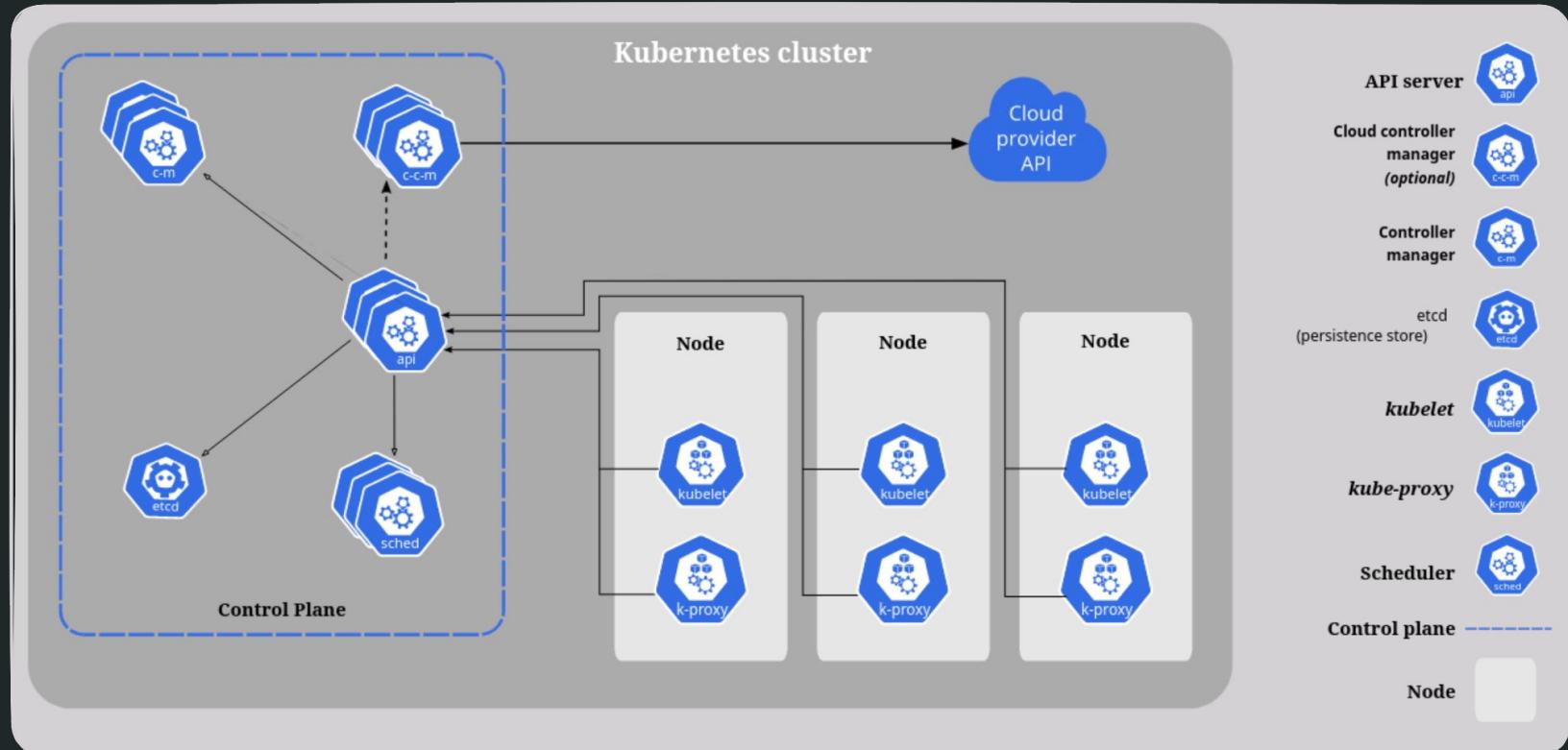
Спасибо за держание в курсе, а зачем это все?



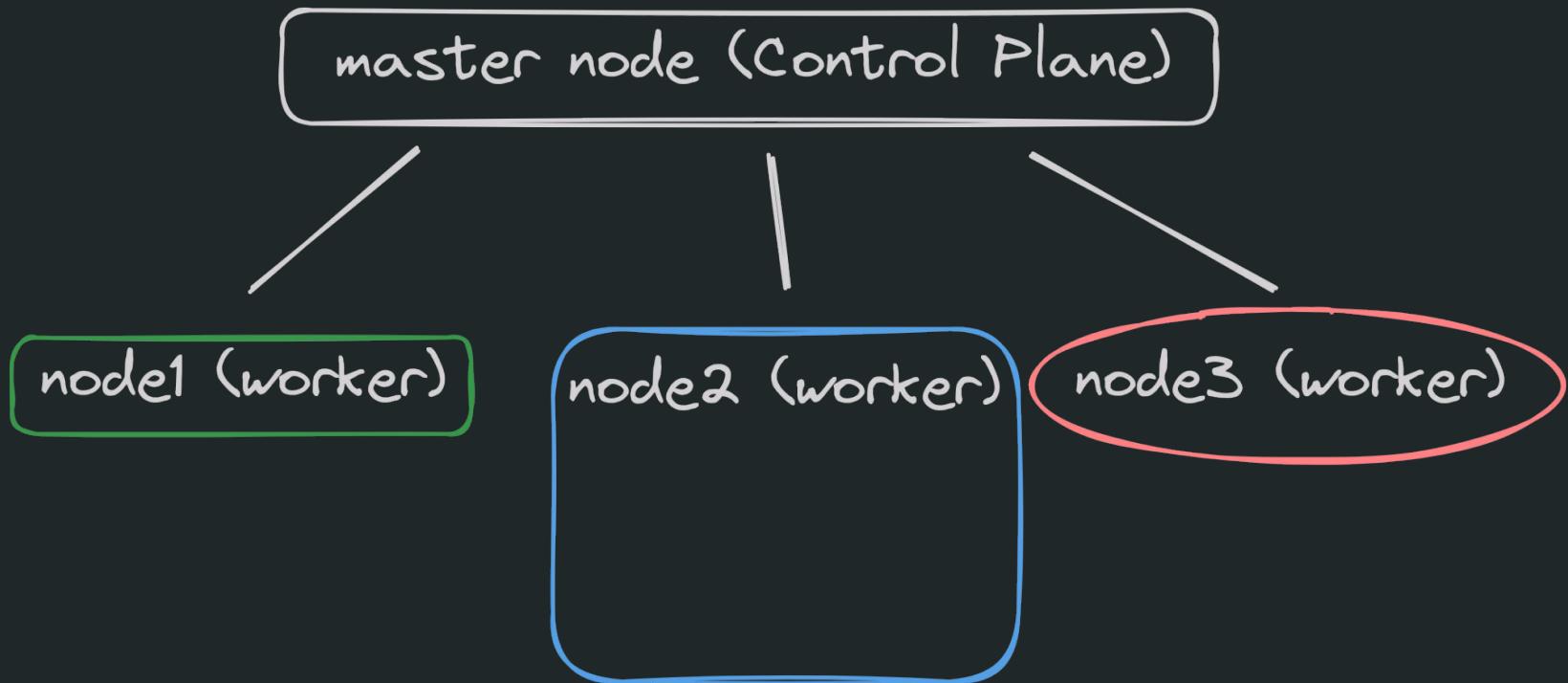
# К архитектуре

---

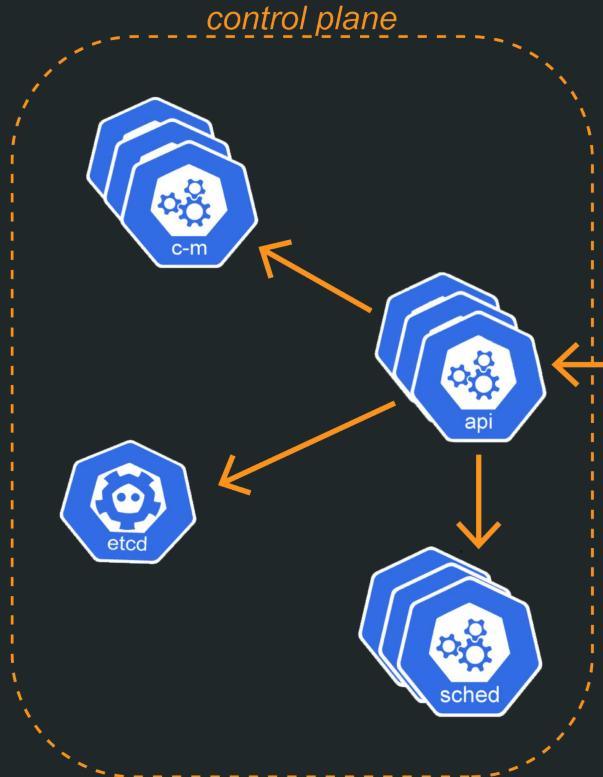
# Kubernetes Components



master/slave, master/worker, master/minion, controller/agent

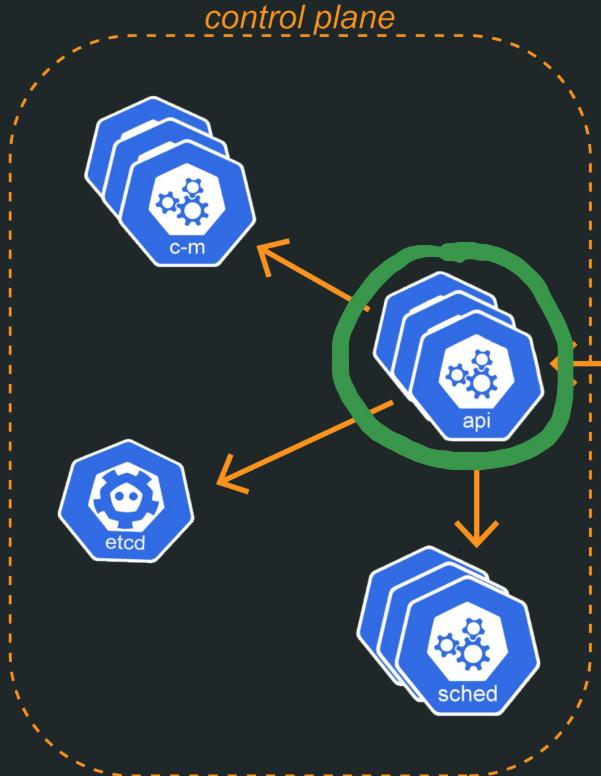


# Control Plane



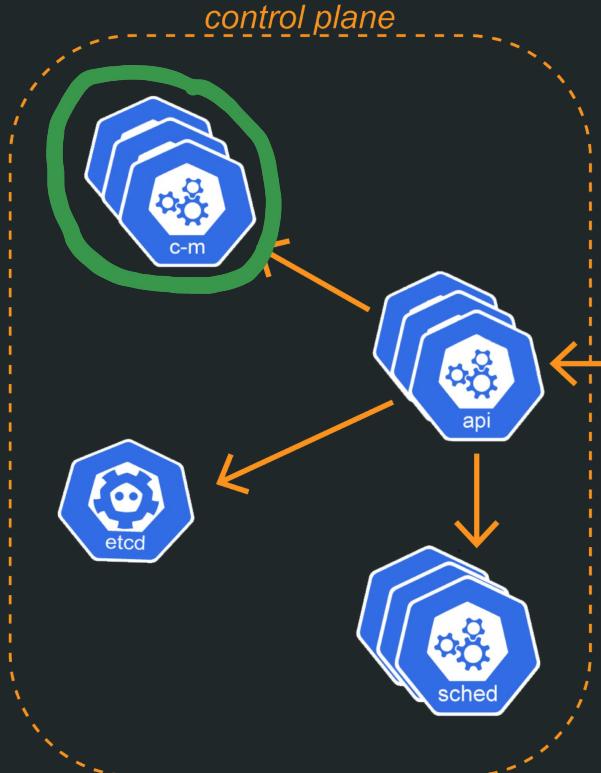
Мозг кластера. Отвечает за логику сущностей, следит за событиями (и реагирует на них), хранит состояние кластера.

# API Server



Интерфейс взаимодействия с Control Plane.  
Используется как самими компонентами  
мастера, так и “внешними” (воркеры, kubectl,  
GUI-админки)

# Controller Manager

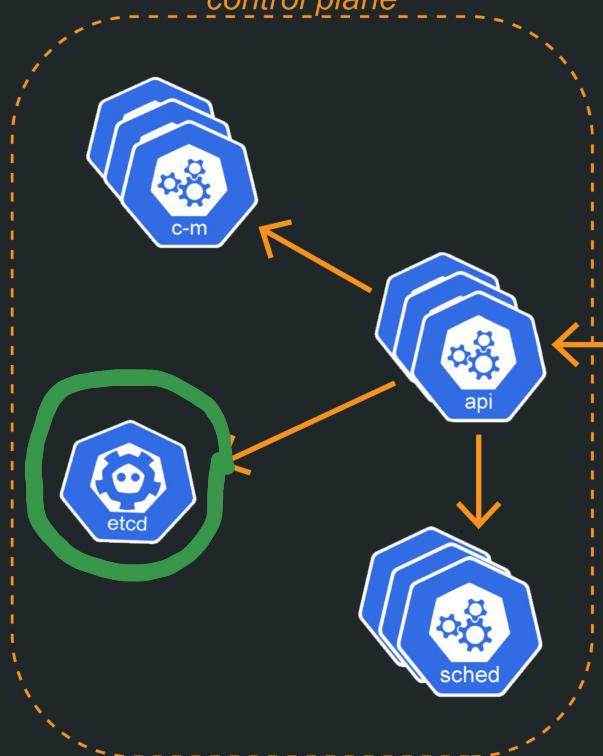


Компонент, запускающий группу контроллеров - каждый из них следит (и вносит соответствующие изменения) за различными аспектами кластера. Среди прочих: Node controller, Deployment controller, Service controller и т.д. - по контроллеру буквально для каждого вида объектов и компонент.

# etcd (powered by

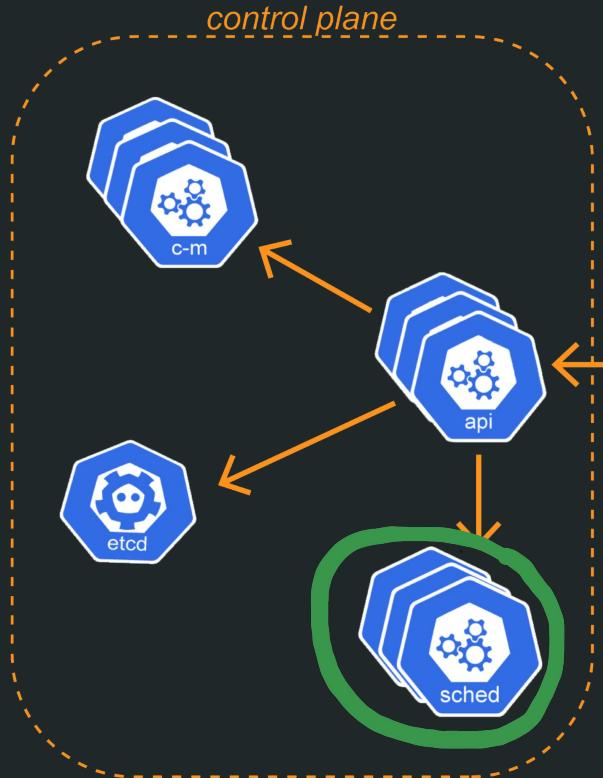


*control plane*



Хранилище данных, информации о компонентах кластера, объектах, состояниях. Выбран как стандарт из-за надежности, отказоустойчивости, эффективности.

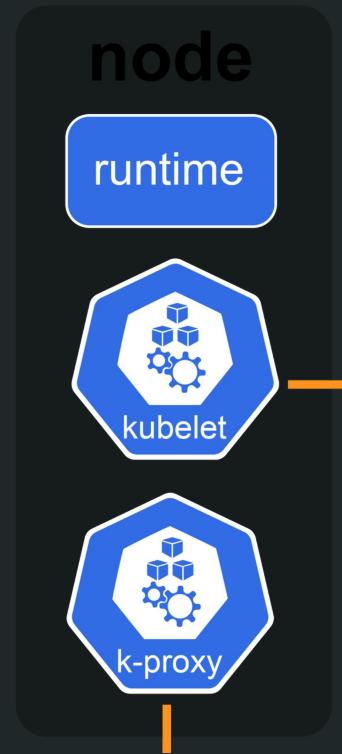
# Scheduler



Своего рода регулировщик - этот компонент решает, на какой ноде должен быть запущен новый под с учетом особенностей пода (запросы и ограничения по ресурсам, affinity) и ноды (“влезет ли”).

# Node

“Рабочая лошадка”. Машина(ы), на которой происходит запуск подов.



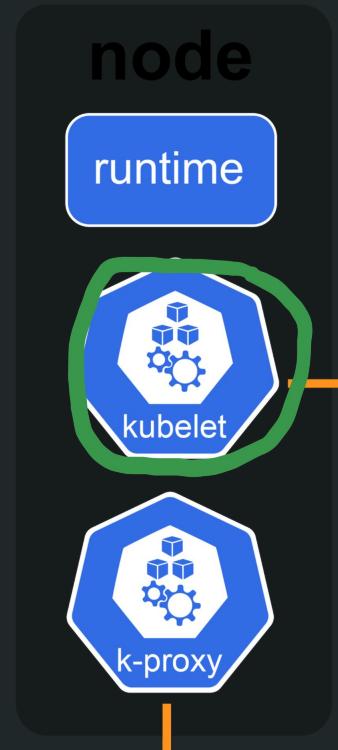
# Container Runtime

Запускает и управляет жизненным циклом контейнеров в привычном для, например, Docker, понимании.



# kubelet

Компонент-демон (агент ноды), который на основе PodSpecs запускает поды и следит за их контейнерами, внося правки при наличии изменений в спеках со стороны Control Plane

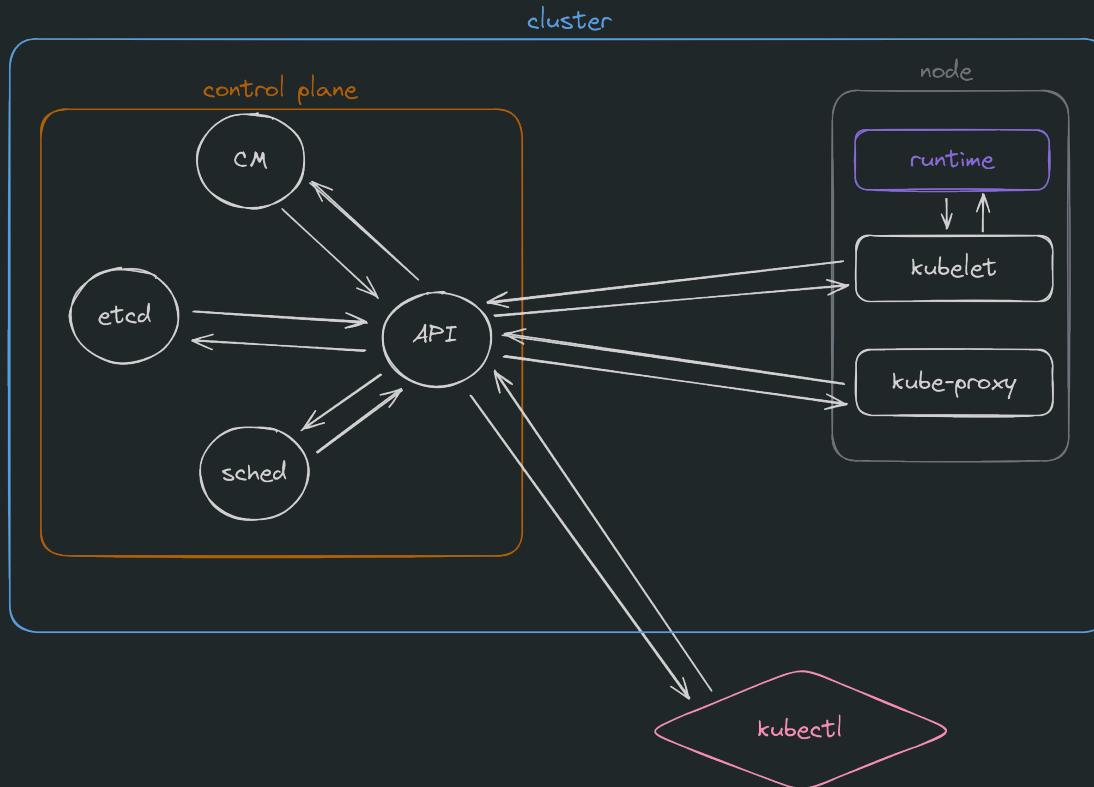


# kube-proxy

Компонент, управляющий сетевыми политиками на ноде - прокси, который выставляет сетевые доступы до подов (и сервисов) по DNS-имени. Может играть роль Load Balancer.



# Пример: Создание пода



# Вопрос для экзамена

Объясните назначение и взаимосвязь компонентов кластера kubernetes (API server, etcd, Scheduler, Controller Manager, kubelet, kube-proxy, Container runtime).

