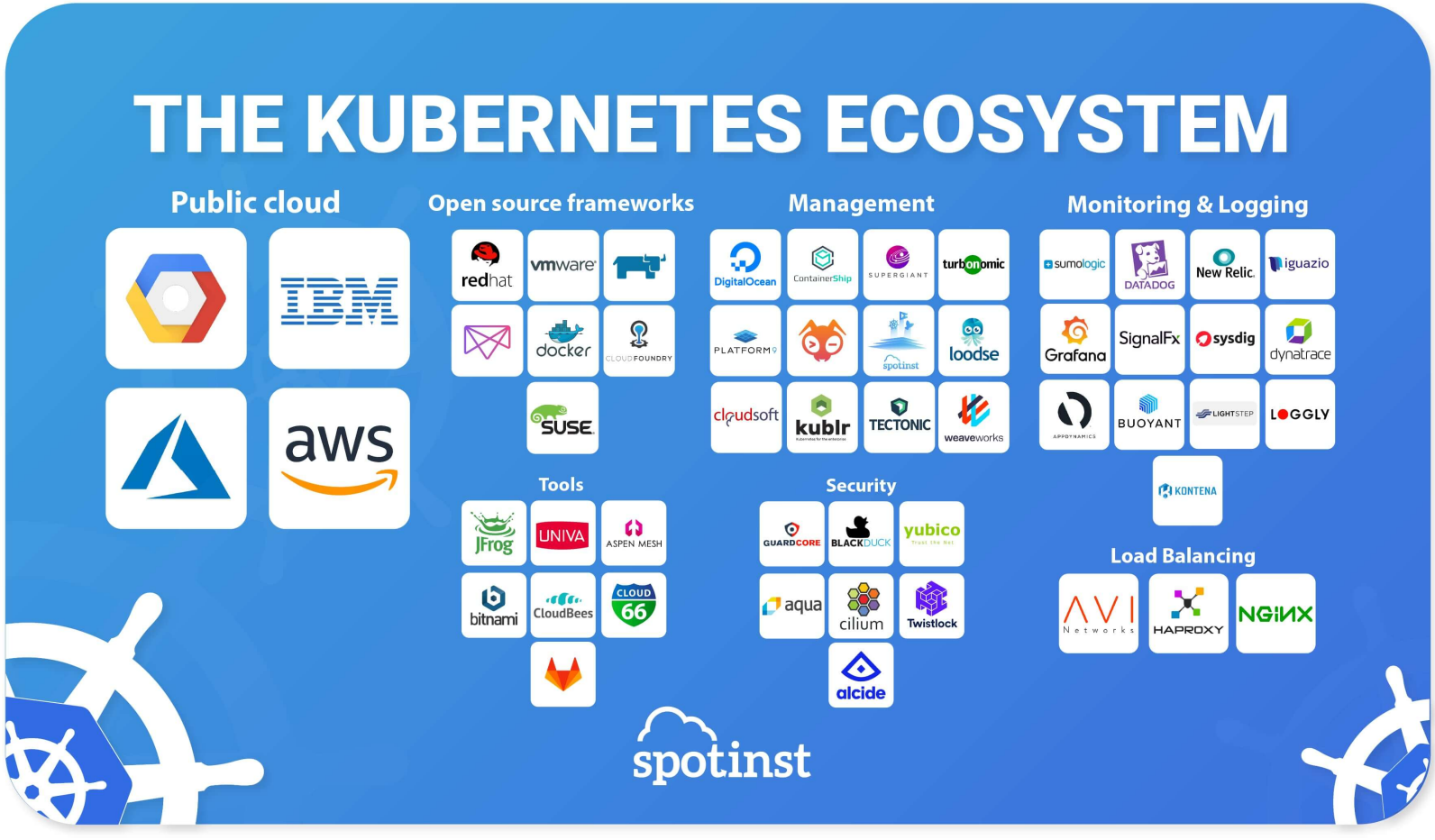

THE KUBERNETES ECOSYSTEM
Public cloud
IBM
aws
Open source frameworks
redhat
vmware
docker
CLOUDFOUNDRY
SUSE
Management
DigitalOcean
ContainerShip
SUPERGIANT
turbonomic
PLATFORM9
spotinst
loodse
cloudsoft
kublr
TECTONIC
weaveworks
Monitoring & Logging
sumologic
DATADOG
New Relic.
iguazio
Grafana
SignalFx
sysdig
dynatrace
BUOYANT
LIGHTSTEP
LOGGLY
KONTENA
Tools
JFrog
UNIVA
ASPEN MESH
bitnami
CloudBees
CLOUD 66
Security
GUARDCORE
BLACKDUCK
yubico
aqua
cilium
Twistlock
alcide
Load Balancing
AVI Networks
HAPROXY
NGINX
spotinst

Не забудь включить запись!

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

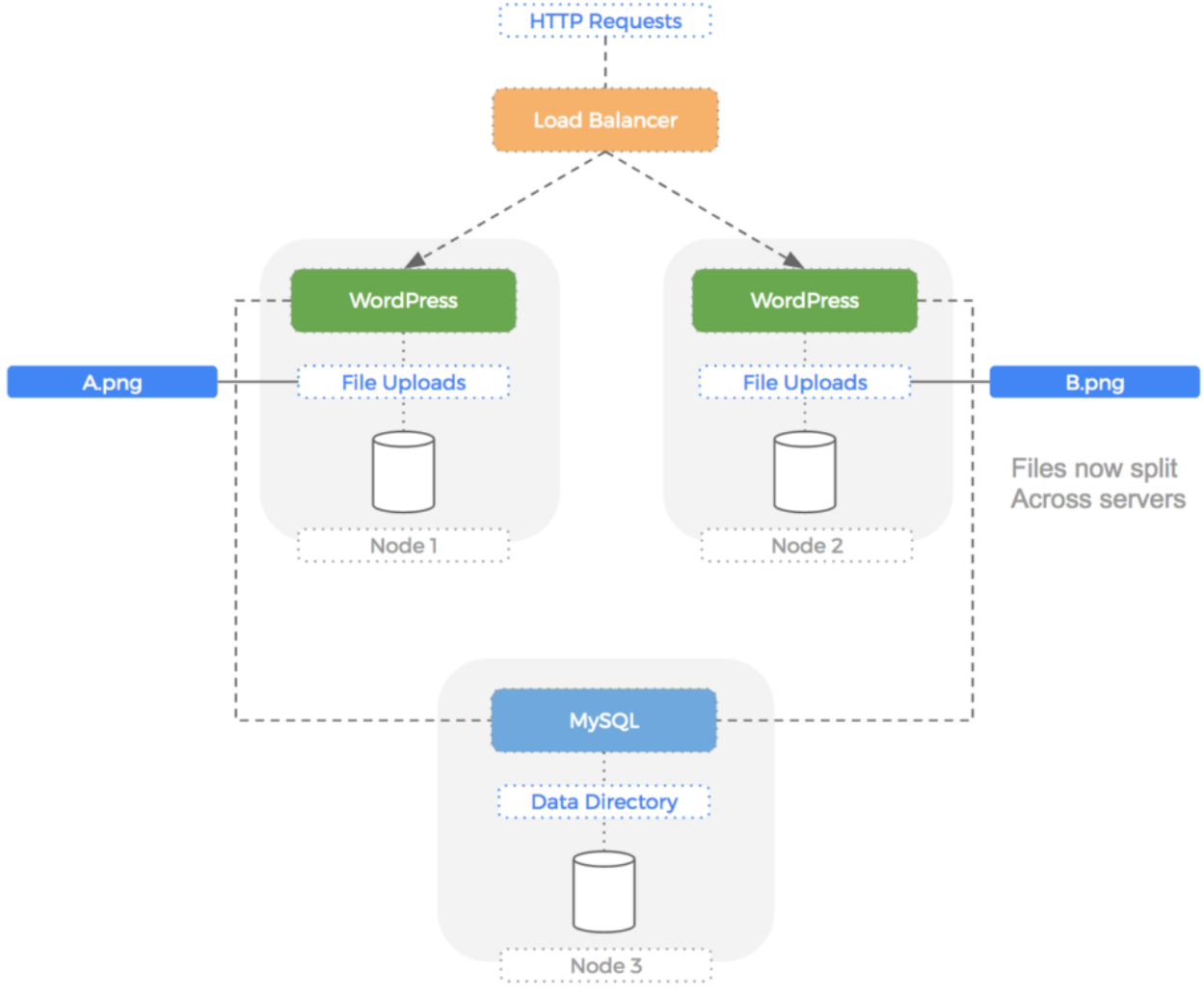
Ставьте + если все хорошо

Lecture 8. Деплой веб-приложений в Kubernetes на примере WordPress

Plan

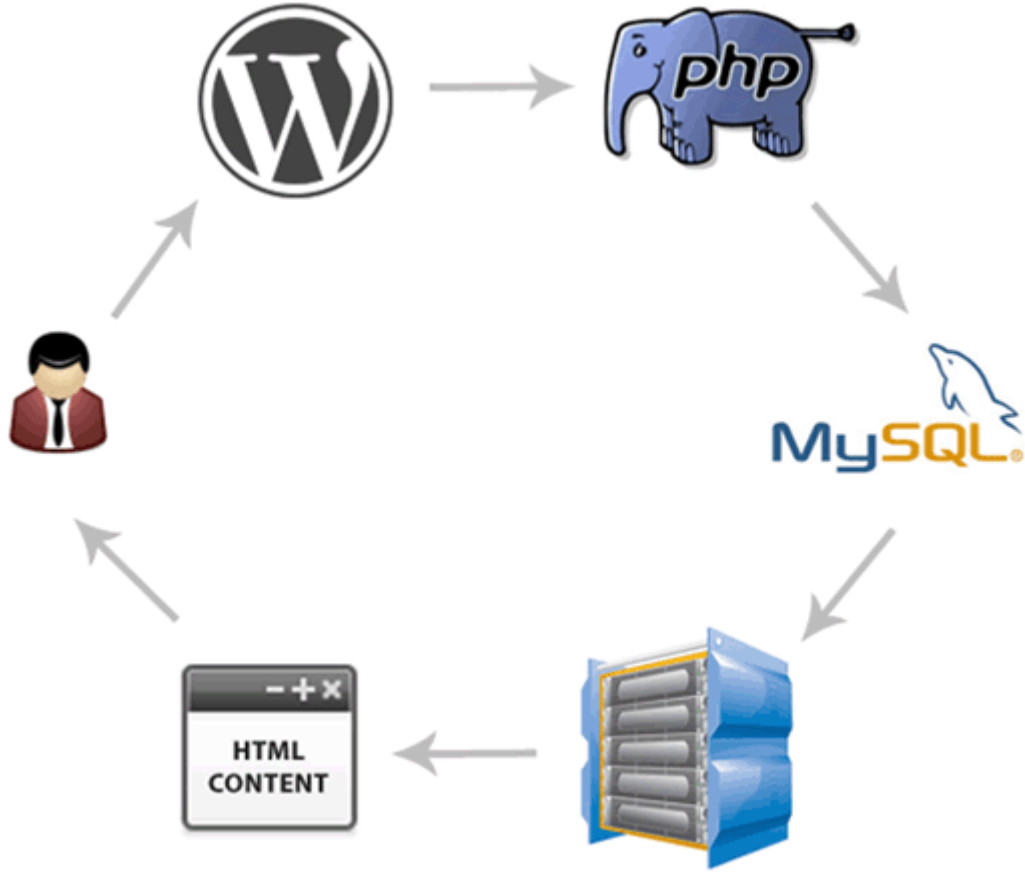
- Цель занятия:
  - Объединить все знания для деплоймента приложения.
  - Посмотреть, как выглядит процесс кубернетизации приложения.
  - Сложить общую картинку в голове о том, зачем всё это необходимо.
- После занятия мы, надеюсь, сможем:
  - Задеплоить код WordPress
  - Развернуть базу данных в kubernetes для блога
  - Опубликовать сервис в сети
  - Поломать minikube
  - Снести всё к чёртовой бабушке
  - Задеплоить блог в кубернетисе для любимой бабушки

Deploy WordPress into Kubernetes

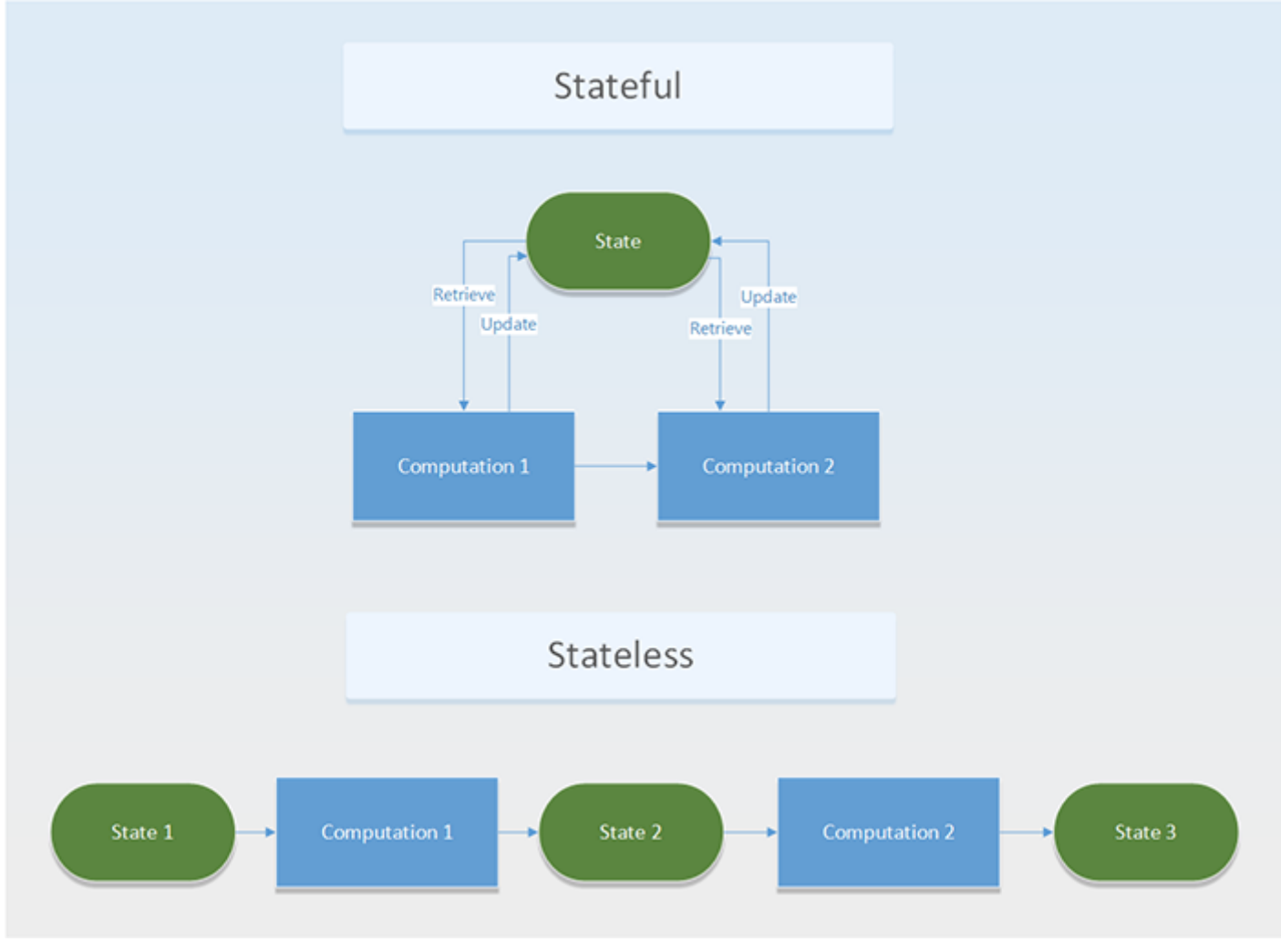


Запускаем Wordpress в minikube.

WordPress + MySQL



Stateless и Statefull приложения



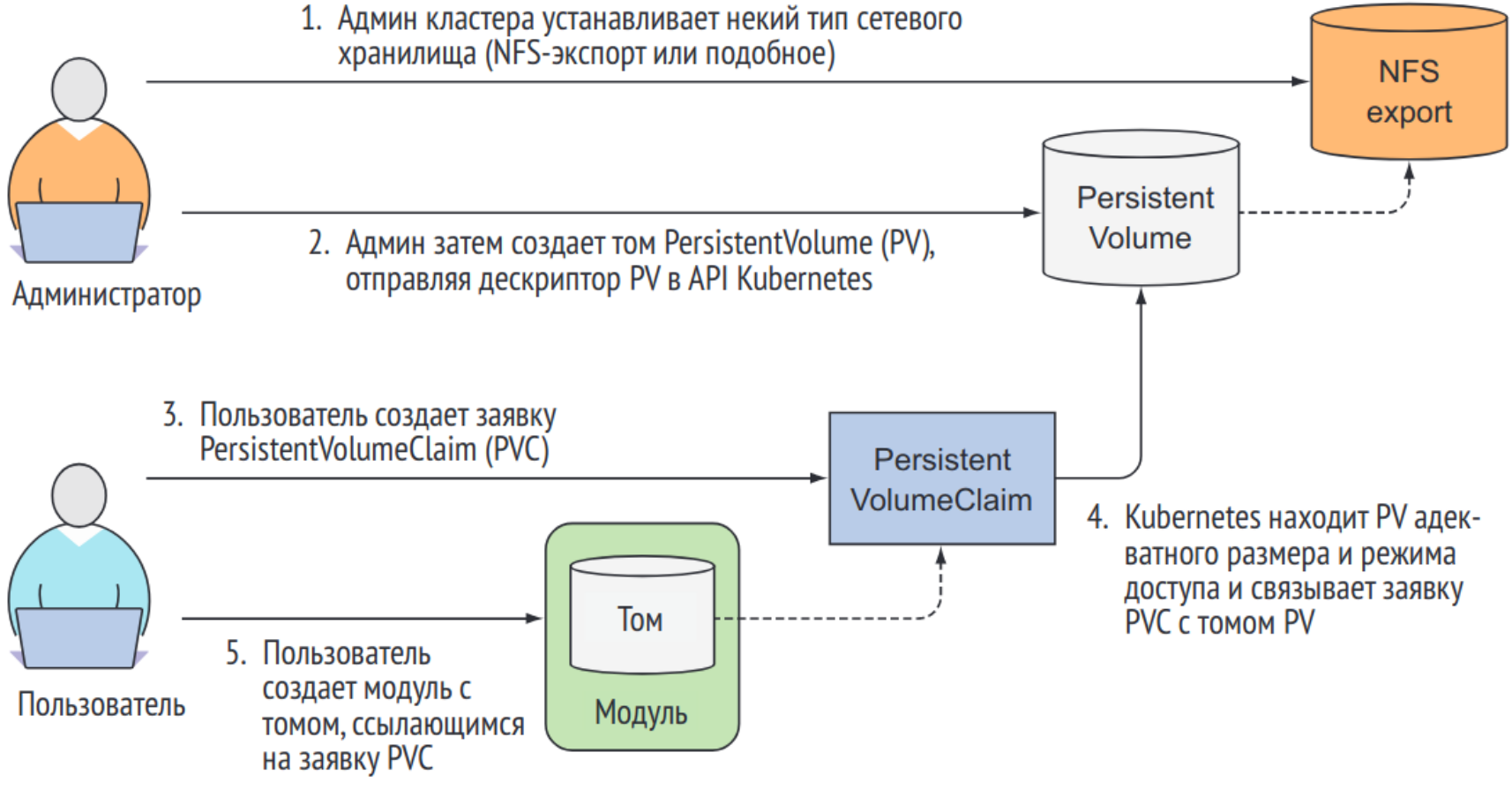
[Ссылка на статью на медиуме]

Какие абстракции нам смогут помочь

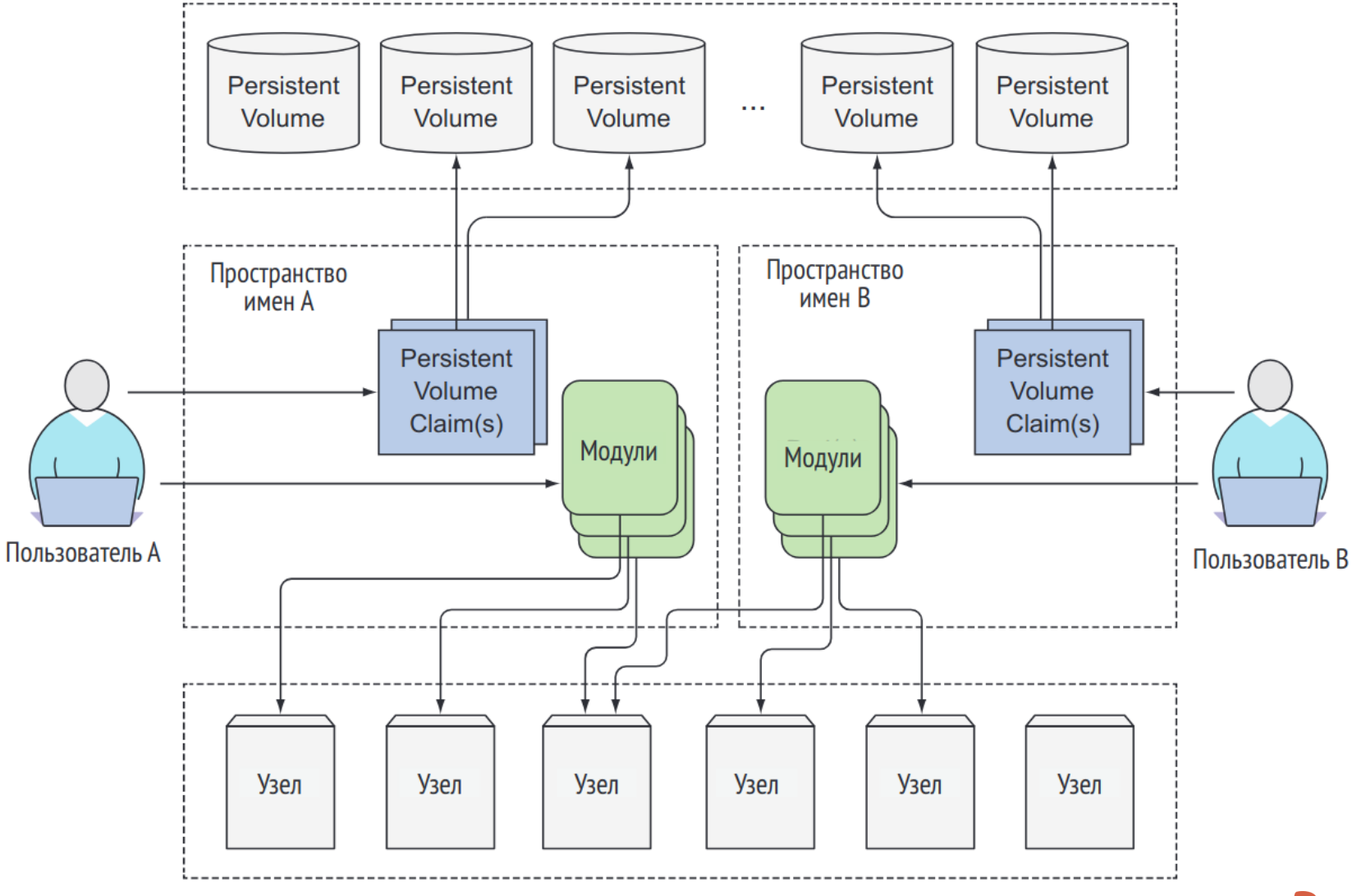
Ресурсы высокого уровня Kubernetes

- **PersistentVolumeClaim**
- **Secret**
- **MySQL Deployment**
- **Deployment для WordPress**
- **Service**

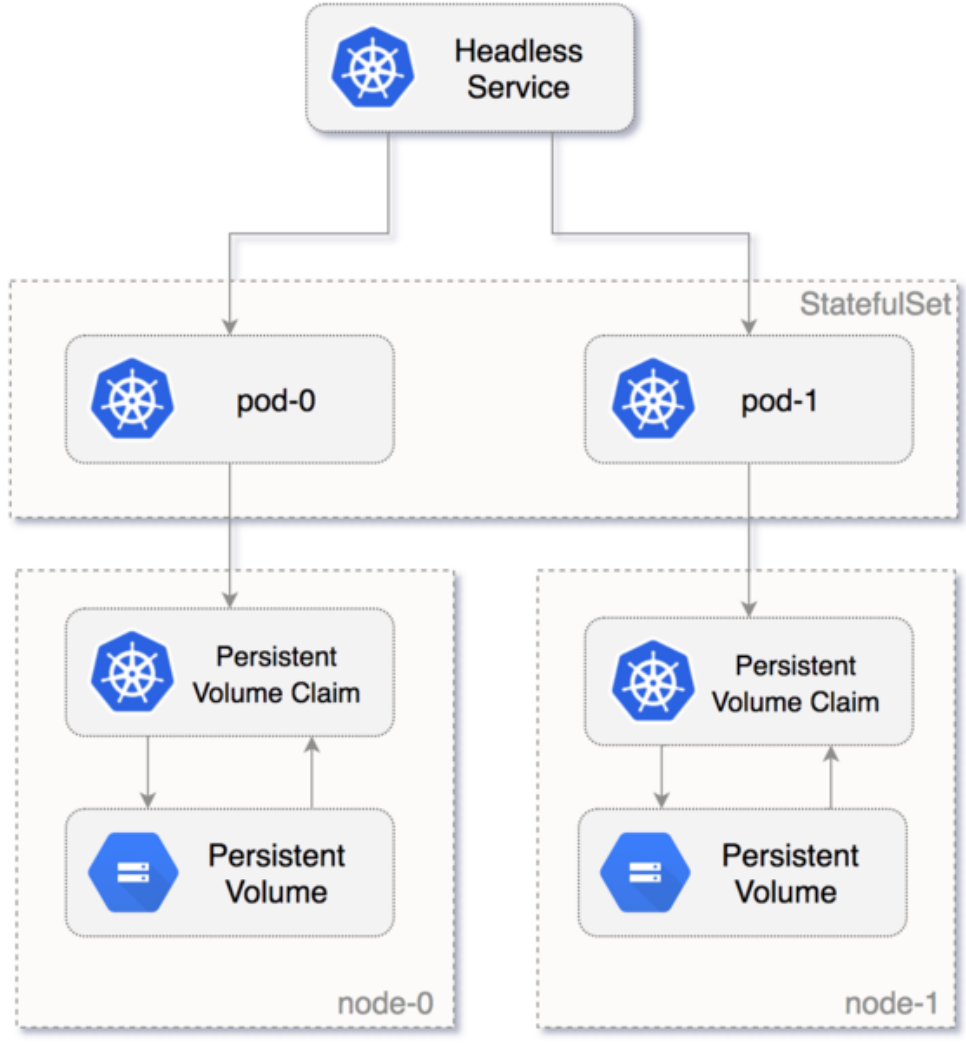
Deployment | Persistent disk



Persistent volume claim



StatefulSet vs Deployment



Demo. Предлагаю вам запустить minikube

Запускаем WordPress

Скопируйте эти строчки из чата и запустите в своём Minikube

```
kubectl run wordpress --image=wordpress:5.2.2-php7.1-apache --port=8080 --
generator=run/v1
kubectl get po
kubectl describe pod
```

Когда `WordPress` запустится, поставьте в чат единичку

Если что-то пойдёт не так, поставьте минус, будем разбираться

Переадресация портов на pod

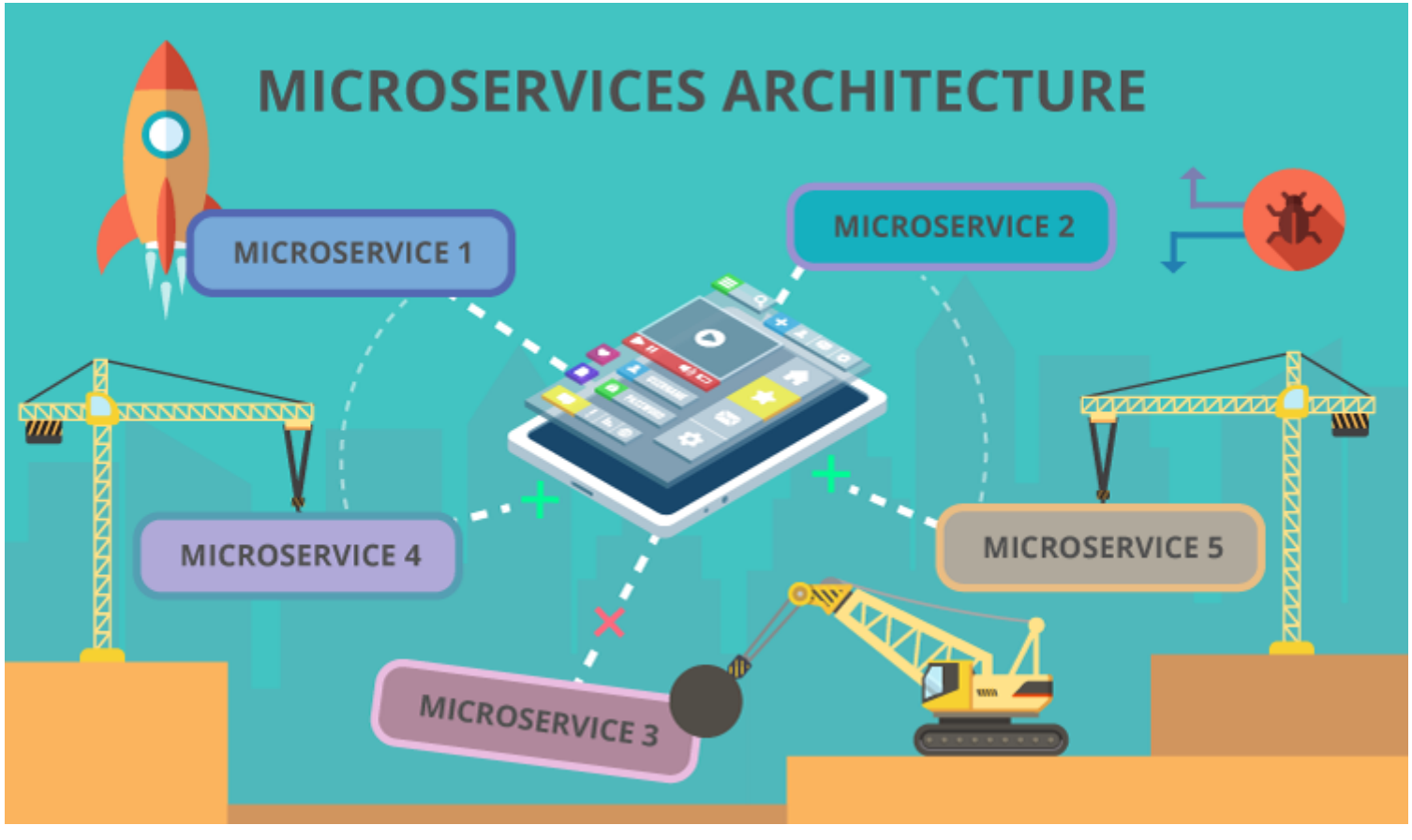
```
kubectl port-forward wordpress-v5z4m 8888:80
```

Запускаем MySQL

```
kubectl run mysql --image=mysql:5.6 --port=3306 --generator=run/v1 --
env=MYSQL_ROOT_PASSWORD=root
kubectl get po
kubectl describe pod
```

Resource	api group	kubectl command
Pod	v1	`kubectl run --generator=run-pod/v1`
Replication controller (deprecated)	v1	`kubectl run --generator=run/v1`
Deployment (deprecated)	extensions/v1beta1	`kubectl run --generator=deployment/v1beta1`
Deployment (deprecated)	apps/v1beta1	`kubectl run --generator=deployment/apps.v1beta1`
Job (deprecated)	batch/v1	`kubectl run --generator=job/v1`
CronJob (deprecated)	batch/v1beta1	`kubectl run --generator=cronjob/v1beta1`
CronJob (deprecated)	batch/v2alpha1	`kubectl run --generator=cronjob/v2alpha1`

Как мы представлем себе работу с микросервисами



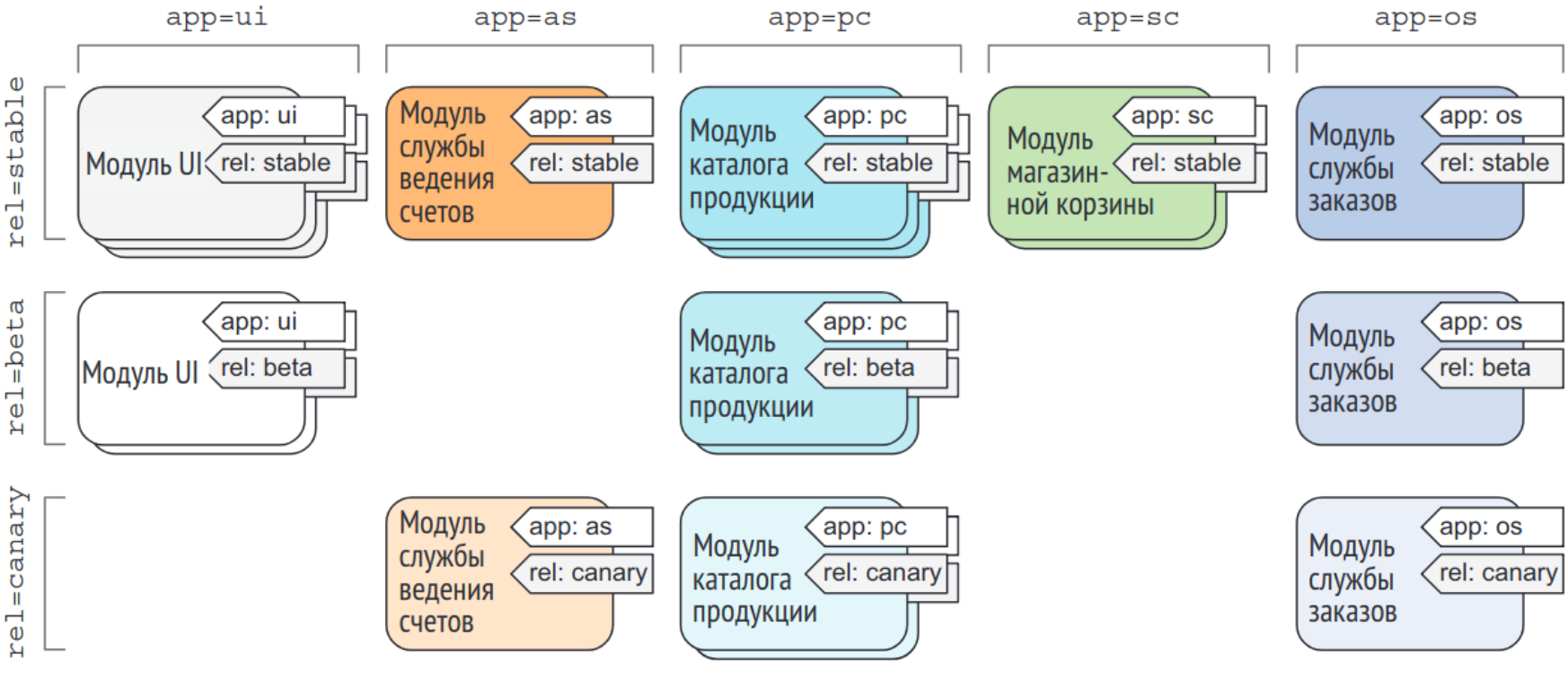
Как они выглядят, когда модули никак не тегированы

Знакомство с метками

Kubernetes Uses Labels

- Can query based on these labels

Организация модулей с помощью меток.



Запустим pod с wordpress с помощью yaml

wordpress-manual.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: wordpress-manual
spec:
  containers:
  - image: wordpress:4.8-apache
    name: wordpress-frontend
    ports:
    - containerPort: 8080
      protocol: TCP
```

Указание меток при создании модуля

wordpress-manual-**with**-labels.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: wordpress-manual-v2
  labels:
    creation_method: manual      # прикрепляем метки к нашему поду
    env: prod
spec:
  containers:
  - image: wordpress:4.8-apache
    name: wordpress-frontend
    ports:
    - containerPort: 8080
      protocol: TCP
```

Запускаем под с WordPress

```
kubectl create -f wordpress-manual-with-labels.yaml
```

Смотрим лабелечки

```
kubectl get po
kubectl get po --show-labels
```

Смотрим только интересующие нас метки

```
kubectl get po -L creation_method,env
```

Изменение меток

```
kubectl label po wordpress-manual creation_method=manual
```

При изменении существующих меток нужно использовать параметр --overwrite.

```
kubectl label po wordpress-v2 env=debug --overwrite
```

```
kubectl get po -L creation_method,env
```

Перечисление подмножеств pod'ов посредством селекторов меток

Селектор меток может выбирать ресурсы в зависимости от того:

- содержит ли (или не содержит) ресурс метку с определенным ключом

```
kubectl get po -l env
```

- содержит ли ресурс метку с определенным ключом и значением

```
kubectl get po -l creation_method=manual
```

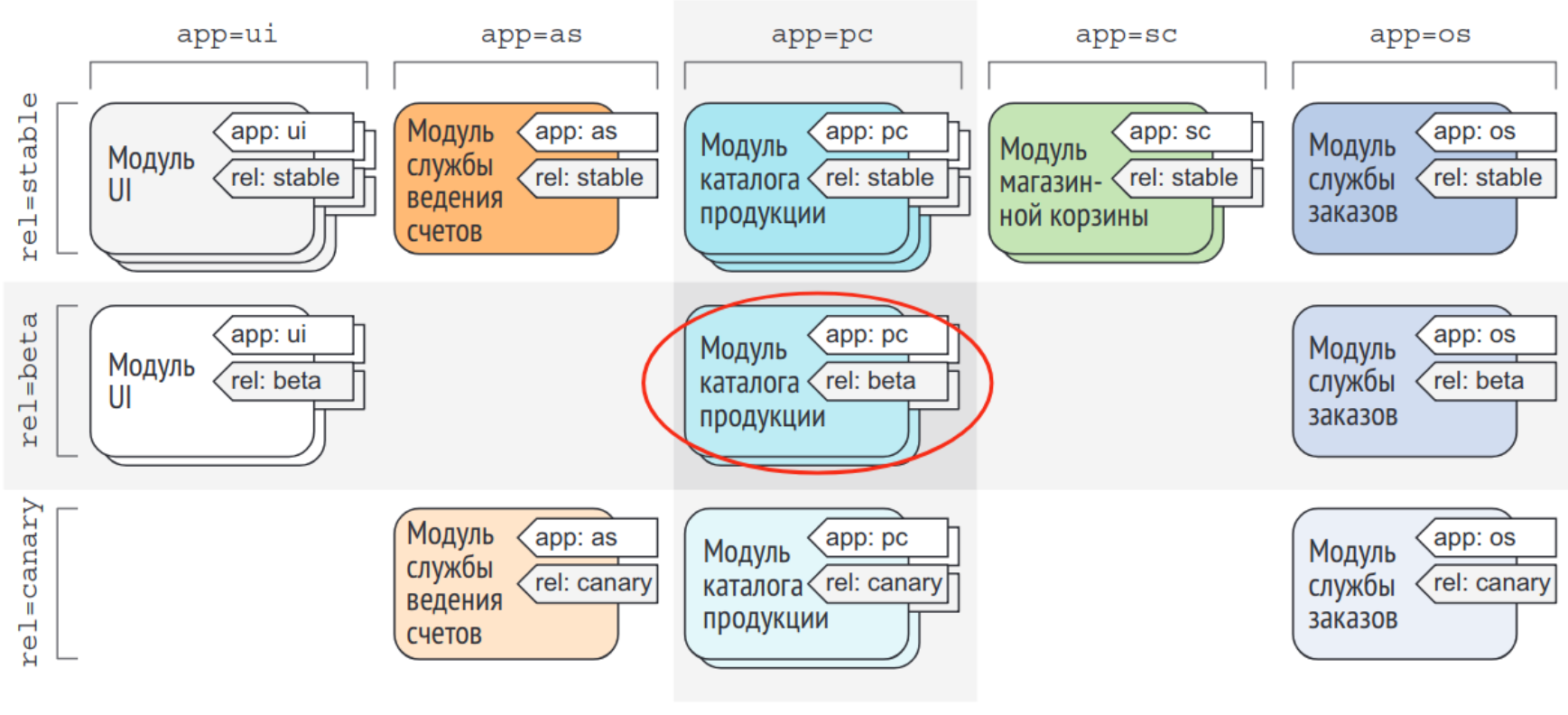
- содержит ли ресурс метку с определенным ключом, но со значением, не равным указанному вами

```
kubectl get po -l '!env'
```

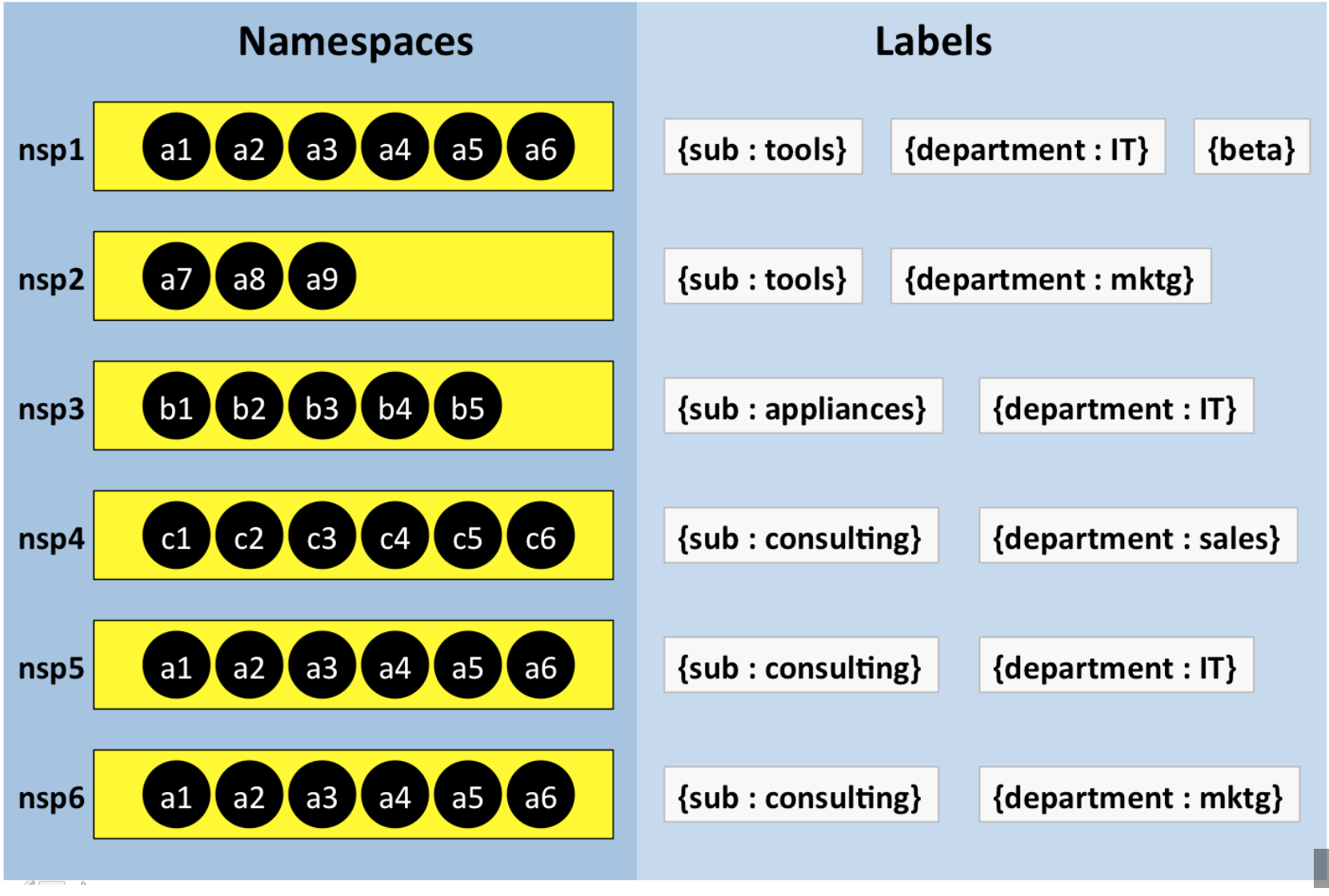
Аналогичным образом можно также сопоставить pod со следующими ниже селекторами меток:

- creation_method!=manual, чтобы выбрать модули с меткой creation_method с любым значением, кроме manual
- env in (prod,devel), чтобы выбрать модули с меткойenv, установлен-ной в prod или development
- env notin (prod,devel), чтобы выбрать модули с меткойenv, установленной в любое значение, кроме prod или devel

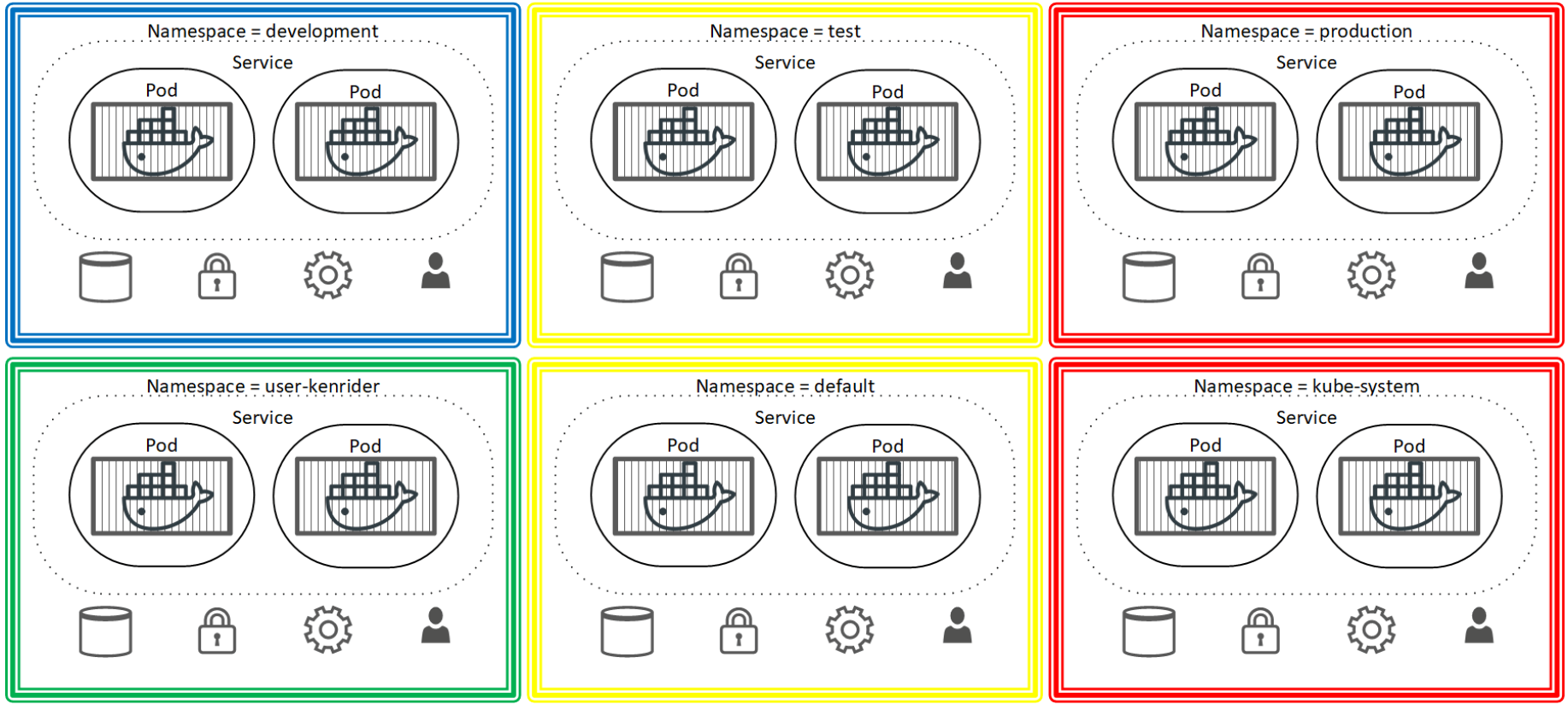
Использование нескольких условий в селекторе меток



Использование пространств имен для группирования ресурсов



Необходимость пространств имен



Создание namespaces

Создание namespace из манифеста YAML

dev-**namespace**.**yaml**

```yaml
apiVersion: v1
kind: Namespace              # создаём пространство имён
metadata:
  name: dev-namespace        # обзываем namespace
```

```
kubectl create -f dev-namespace.yaml
```

Создание namespace из коммандной строки

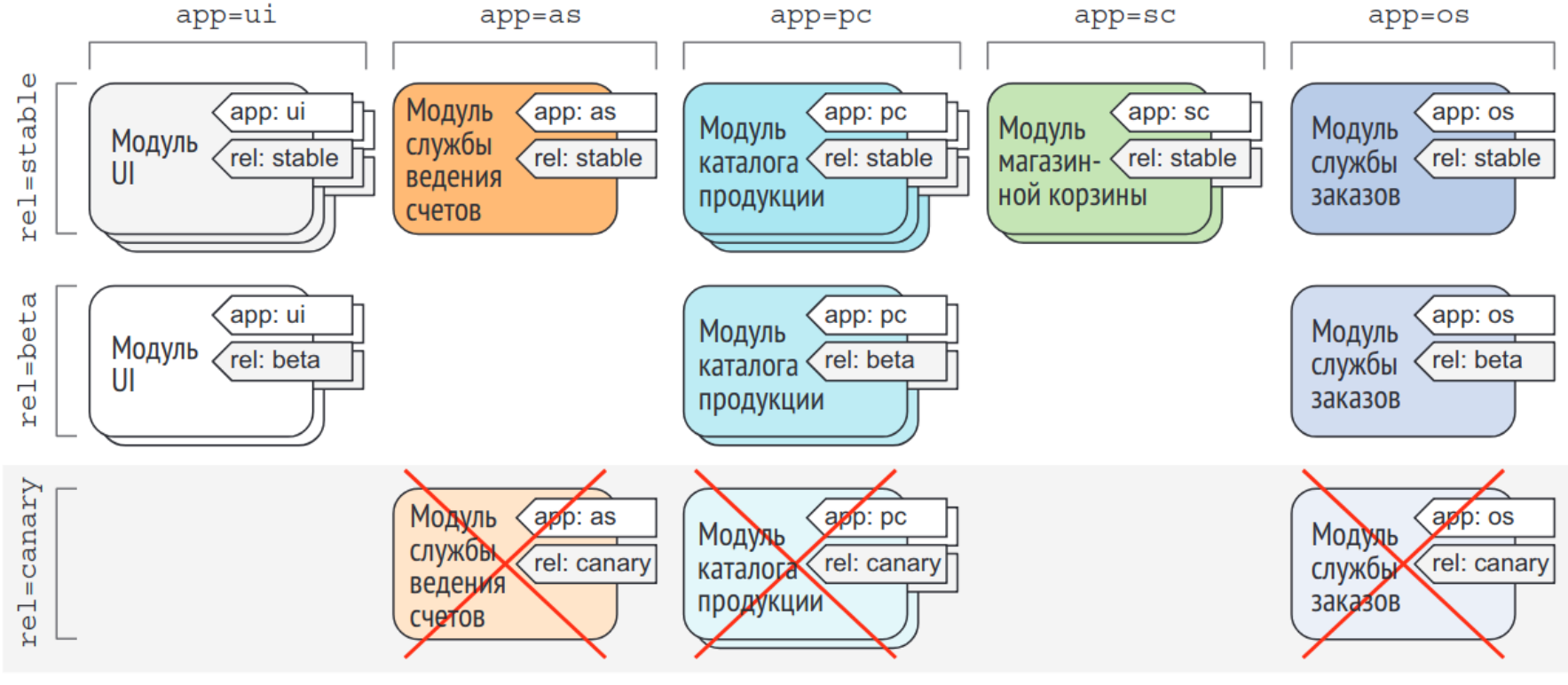
```
kubectl create namespace stage-namespace
```

Запуск pod'ов в пространствах имён

```
kubectl create -f wordpress-manual-with-labels.yaml -n dev-namespace
kubectl create -f wordpress-manual-with-labels.yaml -n stage-namespace
kubectl get po -n dev-namespace --show-labels
kubectl get po -n stage-namespace --show-labels
```

Удаление модулей с помощью селекторов меток

```
kubectl delete po -l creation_method=manual
kubectl delete po -l rel=canary
```



Удаление модулей путем удаления всего пространства имен

```
kubectl delete ns stage-namespace
```

Удаление pod'ов и объектов внутри пространства имён

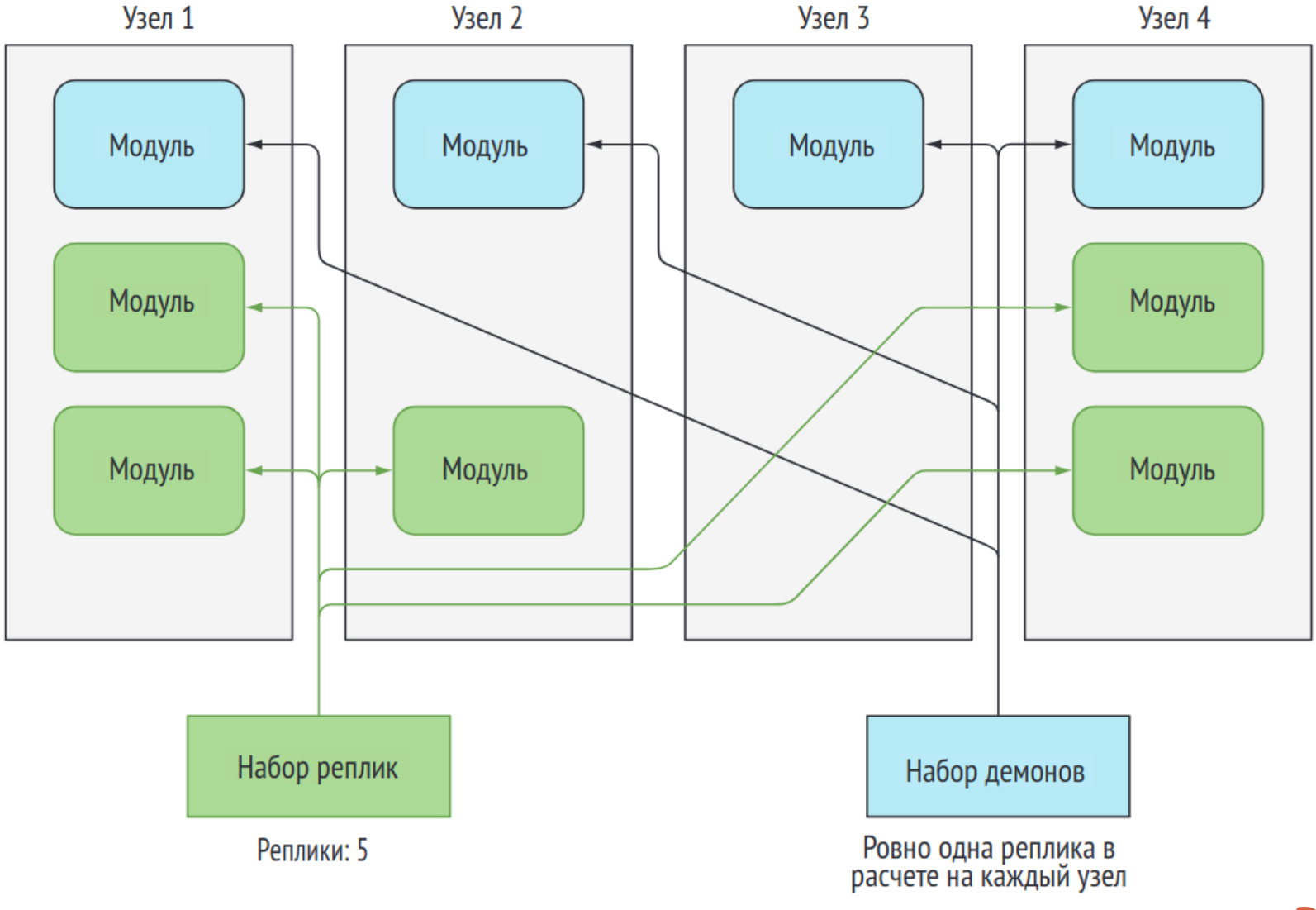
```
kubectl delete po -n dev-namespace --all
kubectl get po -n dev-namespace
```

Удаление (почти) всех ресурсов в пространстве имен

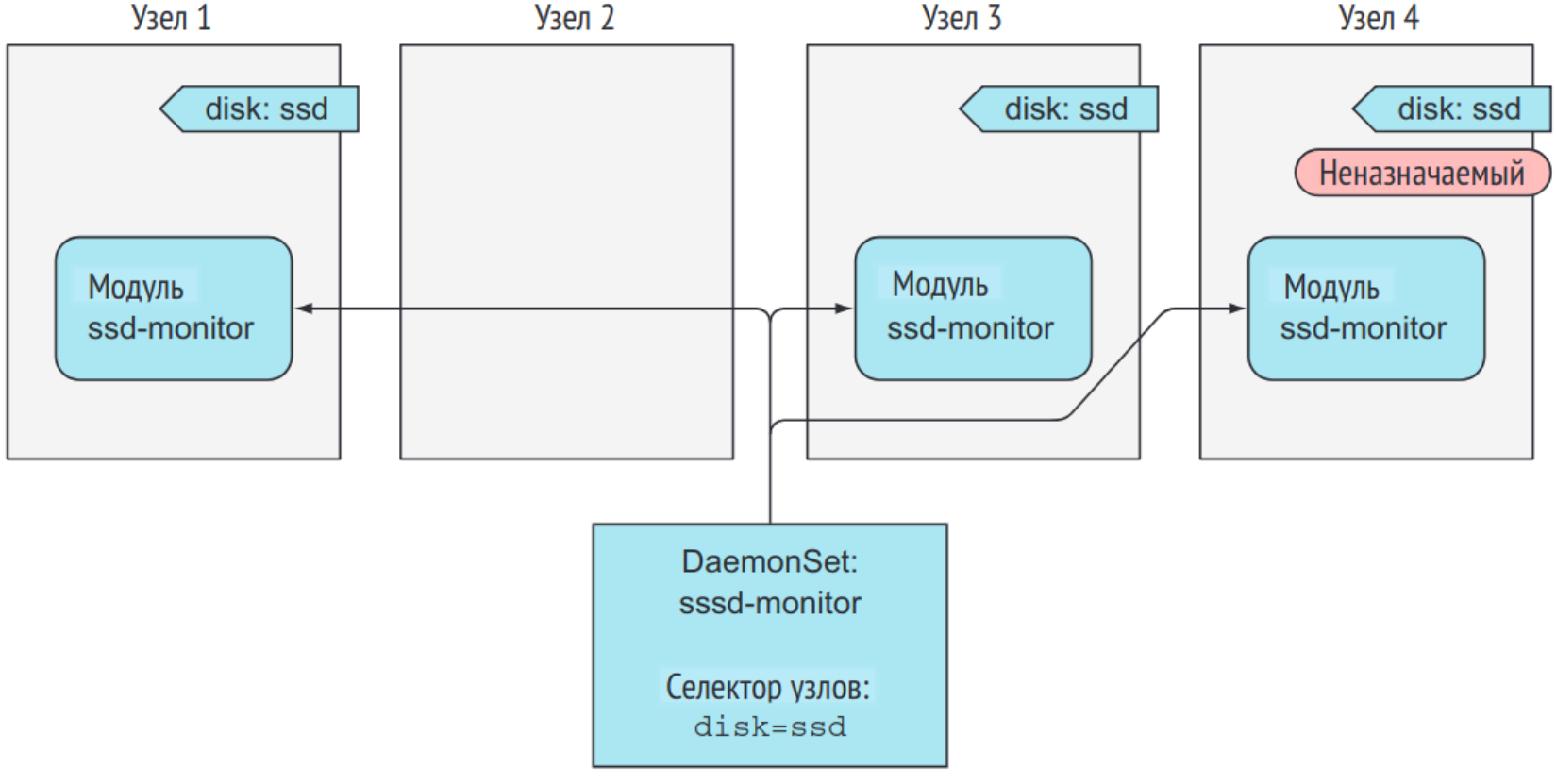
```
kubectl delete all --all
```

ПРИМЕЧАНИЕ. Команда kubectl delete all --all также удаляет службу kubernetes, но она автоматически пересоздастся через некоторое время

DaemonSet


Узел 1
Узел 2
Узел 3
Узел 4
Модуль
Модуль
Модуль
Модуль
Модуль
Модуль
Модуль
Модуль
Модуль
Набор реплик
Набор демонов
Реплики: 5
Ровно одна реплика в расчете на каждый узел

Использование DaemonSet только на определенных узлах



DaemonSet YAML

ssd-monitor-daemonset.yaml

```yaml
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: ssd-monitor
spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
    spec:
      nodeSelector:              # Отбираем ноды с меткой disk=ssd
        disk: ssd
      containers:
      - name: main
        image: luksa/ssd-monitor
```

2. Запускаем это приложение в minikube из манифестов

PersistentVolumeClaim

mysql-volume.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim # выбираем тип запрашиваемого диска
metadata:
  name: mysql-pv-claim # обзываем так, чтобы было понятно, для кого он
  labels:
 app: wordpress # добавляем лабельку, к какому приложению оно принадлежит
spec:
  accessModes:
 - ReadWriteOnce # том может монтироваться на чтение-запись одним узлом
  resources:
 requests:

storage: 250Mi
```

Secret password

mysql-password.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-pass
type: Opaque
stringData:
  password: root # тут мы добавляем желаемый пароль к базуле
```

Deployment

mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress # та же лабелька, что и в PVC
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql # ага, тут мы добавляем лабельку для базульки
  strategy:
    type: Recreate # Все существующие стручки уничтожаются до того, как будут созданы
новые (переведено гуглом)
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
```

Deployment 2

mysql-deployment.yaml

```
spec:
  containers:
    - image: mysql:5.6 # тот контейнер, который мы собираемся задеплоить
      imagePullPolicy: Always # вытягиваем образ всегда, мимо кеша
      name: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD # помните, мы добавляли эту переменную вручную?
          valueFrom:
            secretKeyRef:
              # Import our mysql-pass.password
              # as MYSQL_ROOT_PASSWORD
              name: mysql-pass
              key: password
```

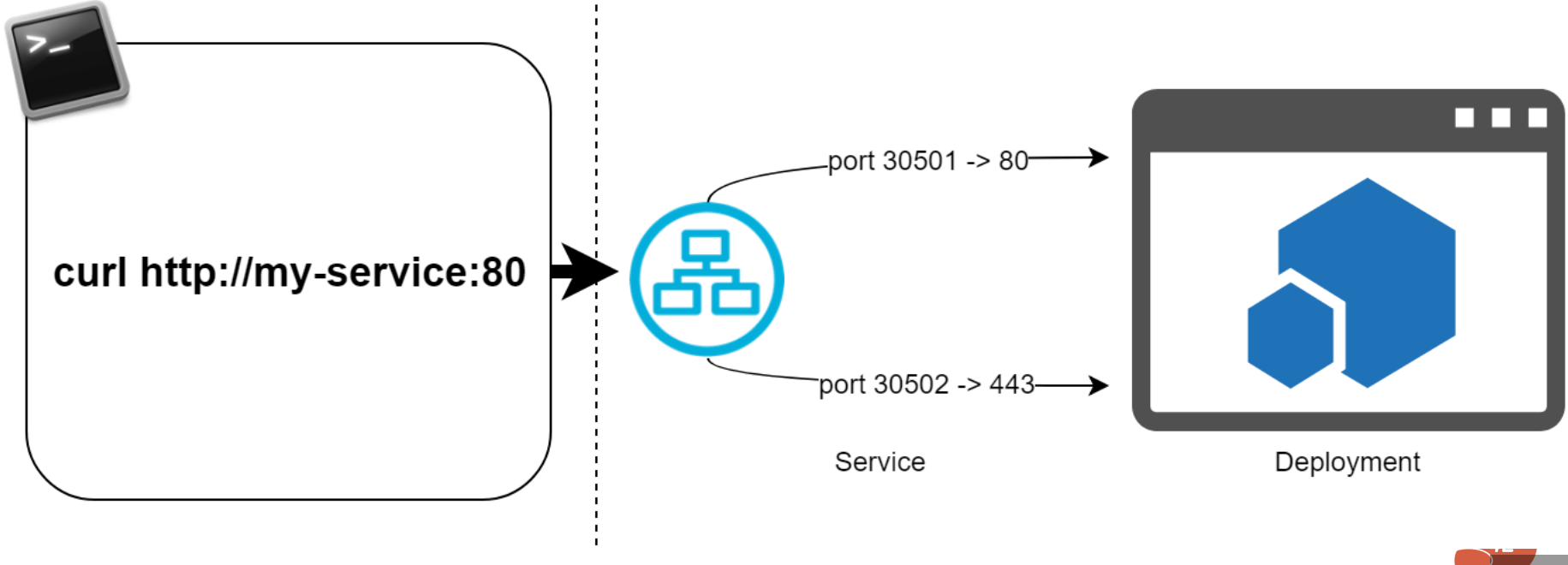
Deployment 3

mysql-deployment.yaml

```
          ports:
            - containerPort: 3306           # на каком порту будем запускаться
              name: mysql
          volumeMounts:
            # Mount our PersistentVolumeClaim
            # to /var/lib/mysql
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql     # прибиваем к внутренней папке внешний
стораж
          resources:
            requests:                       # необходимо для планирования того, на какую
ноду будет деплоиться приложение
              cpu: 100m
              memory: 64Mi
            limits:                         # жёсткое ограничение на под
              cpu: 500m
              memory: 192Mi
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim # и говорим ему, откуда его брать
```

Deployment контролирует, как приложения запускаются, останавливаются и обновляются.

Service контролируют, как клиенты получают доступ к модулю.



Service MySQL

mysql-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:              # обратите внимание, что они соответствуют лабелькам в MySQL
    app: wordpress
    tier: mysql
```

WordPress Deployment 1

wordpress-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
```

WordPress Deployment 2

wordpress-deployment.yaml

```
template:
  metadata:
    labels:
      app: wordpress
      tier: frontend
  spec:
    containers:
    - image: wordpress:4.8-apache
      name: wordpress
      env:
      - name: WORDPRESS_DB_HOST
        value: wordpress-mysql
      - name: WORDPRESS_DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-pass
            key: password
```

WordPress Deployment 3

wordpress-deployment.yaml

```
        ports:
        - containerPort: 80
          name: wordpress
        resources:
          requests:
            cpu: 15m
            memory: 64Mi
          limits:
            cpu: 500m
            memory: 192Mi
```

`

Wordpress service

wordpress-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-http
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: ClusterIP
```

Добавление меток в модули, управляемые deployment'ом

```
spec:
  replicas: 2
```

```
kubectl get pods --show-labels
```

```
kubectl label pod wordpress type=specialpod
kubectl get pods --show-labels
kubectl label pod wordpress-9b99476f-4gvmc app=wordpress-v2 --overwrite
kubectl get po -L app,tier
```

Масштабирование контроллера репликации

```
kubectl scale deployment wordpress --replicas=4
```

Масштабируемся взад-назад


```
kubectl scale deployment wordpress --replicas=2
```

Удаление deployment без удаления pod'ов

```
kubectl delete deployment wordpress --cascade=false
```

Как дотюнивать приложения, чтобы они запускались в кубере

https://12factor.net/ru/

12 factors

I. Единая кодовая база

II. Зависимости

III. Конфигурация

IV. Сторонние службы (Backing Services)

V. Сборка, релиз, выполнение

VI. Процессы

12 factors

VII. Привязка портов (Port binding)

VIII. Параллелизм

IX. Утилизируемость (Disposability)

X. Паритет разработки/работы приложения

XI. Журналирование (Logs)

XII. Задачи администрирования

На этом у нас сегодня всё

Что мы сегодня сделали

- Запустили WordPress и MySQL в minikube из командной строки
- Добавили вручную теги Labels
- Пощупали Replication Controller
- Написали YAML-манифесты для WP и MySQL
  - Request and Limit
    - Memory leak
    - CPU usage

Спасибо
за внимание!