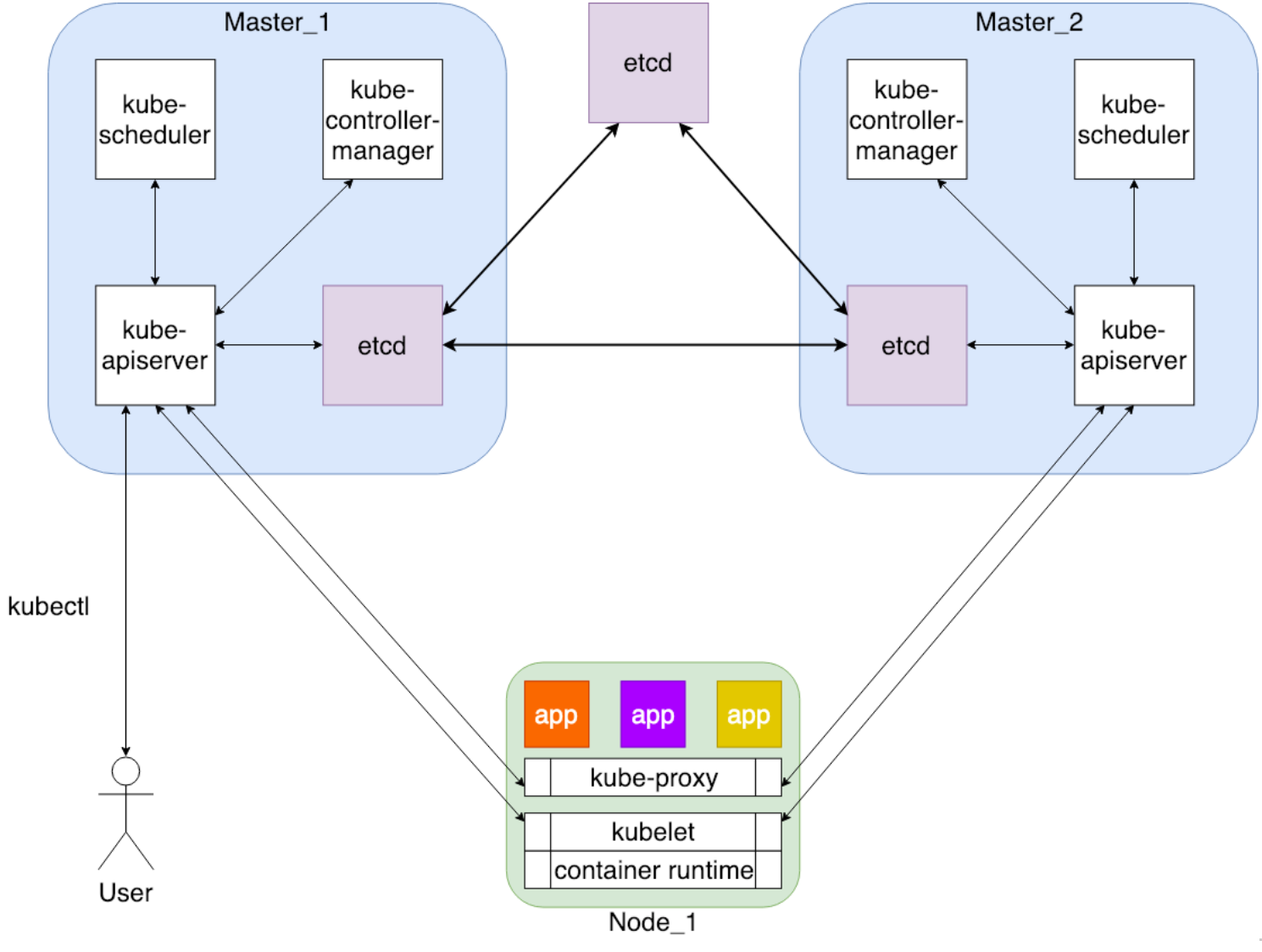
Компоненты Kubernetes

Не забудь включить запись!

План

- Еще один взгляд на архитектуру
- etcd
- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kubelet

Архитектура Kubernetes



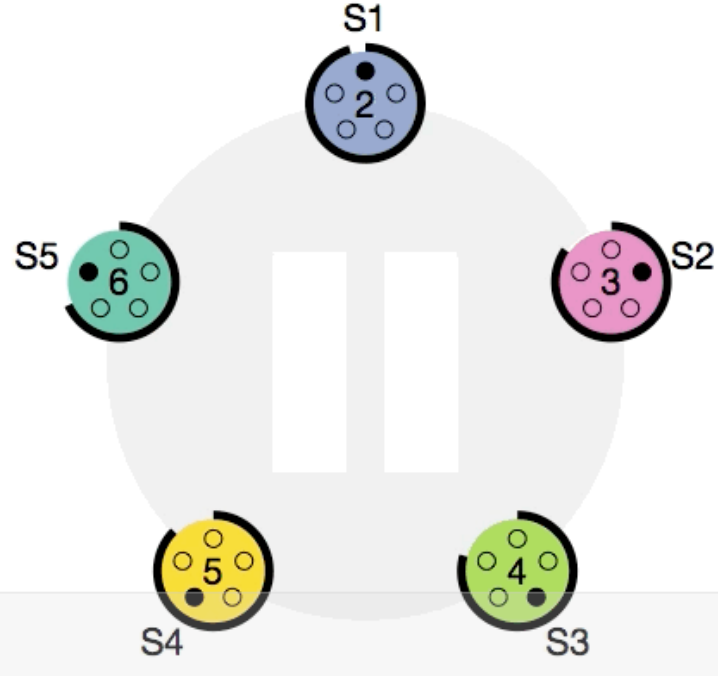
etcd

etcd

- Распределенное key-value хранилище
- Опирается на имплементацию **Raft Consensus Algorithm**

Raft Consensus Algorithm

- Алгоритм сходимости и консистентной репликации лога изменений
- Можно потерять **(N-1)/2** участников (меньшую часть)
- Нечетное количество участников лучше
- В кластере выбирается **leader**, остальные становятся **followers**



https://raft.github.io/

Mail.ru "Базы данных. Алгоритм RAFT"

Кейс: backup и восстановление

~~Идеальный мир~~

- Развертывание кластеров автоматизировано и занимает минимальное количество времени
- Описание всех ресурсов хранится в git в виде манифестов (**IaC**)
- Никто не делает `kubectl run` и прочие ручные действия
- Stateful компоненты:
  - Регулярный backup в системы вне Kubernetes
  - Данные хранятся вне Kubernetes

Нужен ли backup кластера ?

Кейс: backup и восстановление

Реальный мир

Кейс: backup и восстановление

- Описание всех объектов кластера хранится в etcd
- Размер базы данных относительно невелик (поэтому можно быстро и легко сделать ее backup)
- Две опции для резервного копирования:
  - snapshot директории с данными
  - snapshot средствами etcd (`etcdctl`)

Кейс: backup и восстановление

Команда для создания snapshot:

```
etcdctl snapshot save <filename>
```

Нужно помнить про:

- ENV переменную для указания версии etcd (**ETCDCTL_API=3**)
- сертификаты для подключения к etcd (ca, key, certificate)

Кейс: backup и восстановление

Команда для восстановления из snapshot:

```
etcdctl snapshot restore <filename>
```

Особенности `restore`:

- Snapshot не заливается в работающий etcd
- Из snapshot создается новая директория
- Ключом `--data-dir=` можно указать процессу etcd где лежат восстановленные данные

Кейс: backup и восстановление

Нюансы

- Kubernetes - очень быстро изменяющаяся среда. Есть шанс получить неактуальный state
- Возможны некоторые артефакты после восстановления из snapshot
- Надо помнить про сертификаты
- backup etcd никак не касается данных в PV (но хранит информацию о них)
- Как быть с Load Balancer's?

Кейс: backup и восстановление

Что делать:

- Вспомнить *идеальный мир*
- Достигнуть высокой доступности путем создания нескольких реплик etcd
- Административно ограничить права на доступ к etcd
- По возможности не удалять ключи из etcd руками © Zalando
- Отделить etcd для Kubernetes от других кластеров etcd
- Backup etcd

Failure Tolerance and Majority

Четное количество etcd нод

- Не дает дополнительной отказоустойчивости (**Failure Tolerance**)
- Повышает риск отсутствия кворума (**Majority**) при сетевых проблемах

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

Кейс: backup и восстановление

- По умолчанию через каждые 100000 записей snapshot сбрасывается из RAM на диск. Частота конфигурируется:
  - Ключом `--snapshot-count`
  - ENV переменной `ETCD_SNAPSHOT_COUNT`

Кейс: backup и восстановление

Velero

- Утилита для снятия backup и восстановления информации о кластере
- Умеет делать резервное копирование:
  - Ресурсов кластера
  - PV

Кейс: планирование нагрузки

Рекомендации по конфигурации выделенных нод для etcd

Small cluster

A small cluster serves fewer than 100 clients, fewer than 200 of requests per second, and stores no more than 100MB of data.

Example application workload: A 50-node Kubernetes cluster

Provider	Type	vCPUs	Memory (GB)	Max concurrent IOPS	Disk bandwidth (MB/s)
AWS	m4.large	2	8	3600	56.25
GCE	n1-standard-2 + 50GB PD SSD	2	7.5	1500	25

Кейс: масштабирование

С увеличением нод в кластере etcd **увеличивается**:

- Отказоустойчивость
- Производительность на чтение (можно читать с любой ноды)

С увеличением нод в кластере etcd **уменьшается**:

- Производительность на запись (любая запись в кластер реплицируется между всеми нодами)

Оптимальный размер кластера etcd - **5 нод** (by Google).

Масштабирование | ограничения etcd

- **Request size limit** (`1.5 MiB`) - максимальный размер запроса (пары key:value) по умолчанию. Конфигурируется:
  - Ключом `--max-request-bytes`
  - ENV переменной `ETCD_MAX_REQUEST_BYTES`
- **Storage size limit** (`2GB`) - максимальный размер базы данных по умолчанию. Конфигурируется:
  - Ключом `--quota-backend-bytes`
  - ENV переменной `ETCD_QUOTA_BACKEND_BYTES`

> Как правило, проблемы с размером БД начинаются на 1000+ нодах в кластере.

Кейс: георасределение

- Повышает отказоустойчивость
- Успешность зависит от latency и пропускной способности сети между ДЦ
- В случае высокого latency требует тюнинга etcd (без него возможны частые перевыборы лидера и нестабильная работа кластера)
- Стоит помнить про Raft

Tuning

Основные источники проблем:

- Сетевые задержки
- Медленные диски

Методы улучшения ситуации:

- Приоритизация использования сетевых ресурсов (**Traffic Control**)
- Приоритизация использования диска (**ionice**)
- Тюнинг количества событий необходимых для создания снапшота (**ETCD_SNAPSHOT_COUNT**)

Tuning: параметры

- **Heartbeat Interval** (`100ms`) - частота с которой leader рассылает подтверждение о своем лидерстве
- **Election Timeout** (`1000ms`) - время, через которое follower попытается стать leader если не получит heartbeat

Heartbeat Interval рекомендуется делать равным (0.5x - 1.5x) RTT между нодами

- Низкий Heartbeat Interval - увеличение нагрузки на CPU/сеть
- Высокий Heartbeat Interval - замедление реакции на отказы

etcdadm

- Упрощает управление кластером etcd (аналог kubeadm для etcd)
- Заботится о сертификатах
- Поддерживает init кластера из snapshot

Создать кластер:

```
etcdadm init
```

Присоединиться к кластеру:

```
etcdadm join
```

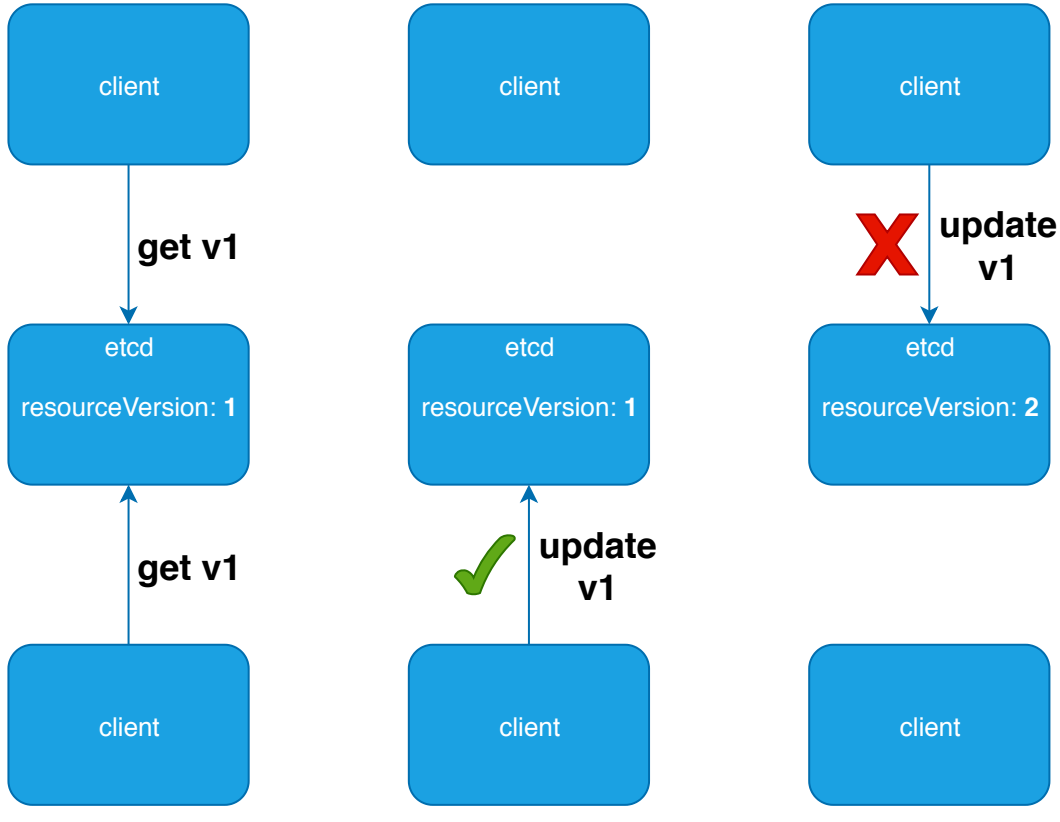
Удалиться из кластера:

```
etcdadm reset
```

Optimistic Locking

resourceVersion

Способ блокировки описания ресурса в etcd при его изменении



Демо. Снимаем резервную копию etcd и восстанавливаемся из нее

kube-apiserver

Терминология

- **group** - набор связанного функционала, например, группа `apps` содержит описание ресурсов `Deployment`, `DaemonSet`, `StatefulSet`, группа `batch` - `Job` и `CronJob`
- **version** - каждая группа API может иметь несколько версий, например описание `CronJob` доступно в версии `v1beta1`, в то время, когда описание `Job` доступно в версии `v1` группы `batch` (один объект может иметь несколько версий, например - `Deployment` находится в `apps/v1beta1`, `apps/v1beta2`, `apps/v1`, `extensions/v1beta1`)
- **kind** - название схемы объекта в API
- **resource** - представление объекта, отправленного или полученного в виде JSON

kubectl

Получить список ресурсов API:

```
kubectl api-resources
```

Получить описание определенного ресурса:

```
kubectl explain <resource> [--recursive]
```

Получить описание полей ресурса, например:

```
kubectl explain pod.spec
```

Версионирование API

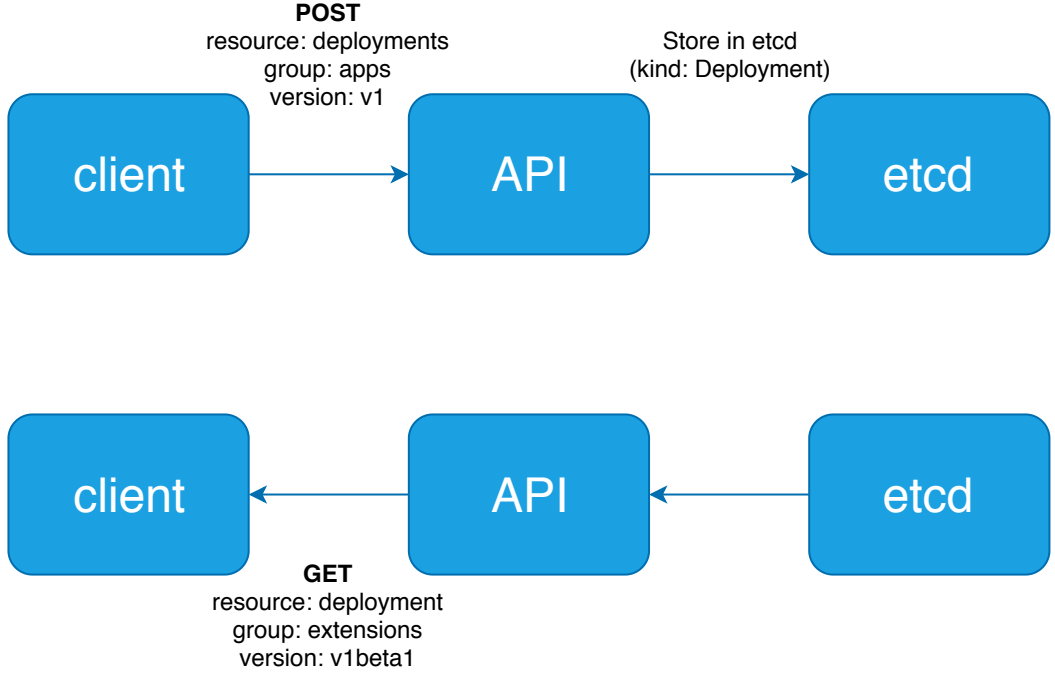
- **v1alpha1** (функционал выключен по умолчанию)
- **v1beta1** (функционал включен по умолчанию)
- **v1** (Stable)

Объект созданный из манифеста с указанием одной версии может быть возвращен в формате любой поддерживаемой версии (`lossless conversion` в API).

Для этого при преобразовании используется дополнительная, внутренняя версия объекта (`internal`), содержащая все поля всех поддерживаемых версий.

`v1beta1 ⇒ internal ⇒ v1`

Storage flow



`--storage-media-type` - какой тип сериализации будет использоваться в etcd (по умолчанию protobuf)

Пример | batch API group

```
{
    "kind": "APIGroup",
    "apiVersion": "v1",
    "name": "batch",
    "versions": [
        {
            "groupVersion": "batch/v1",
            "version": "v1"
        },
        {
            "groupVersion": "batch/v1beta1",
            "version": "v1beta1"
        }
    ],
    "preferredVersion": {
        "groupVersion": "batch/v1",
        "version": "v1"
    }
}
```

Версионирование API

Общие рекомендации

- Внимательно читать Release Notes и блог Kubernetes, например: https://kubernetes.io/blog/2019/07/18/api-deprecations-in-1-16/
- Мигрировать объекты в максимально актуальную версию **перед** обновлением кластера

Версионирование API

Если не следовать общим рекомендациям

- Старые манифесты могут стать неработоспособными после обновления
- Возможна ситуация когда слишком старый объект невозможно десериализовать из JSON/protobuf

> Может выстрелить если версия объекта, которая на момент его создания считалась наиболее актуальной, в новой версии Kubernetes объявлена deprecated и ее схема удалена

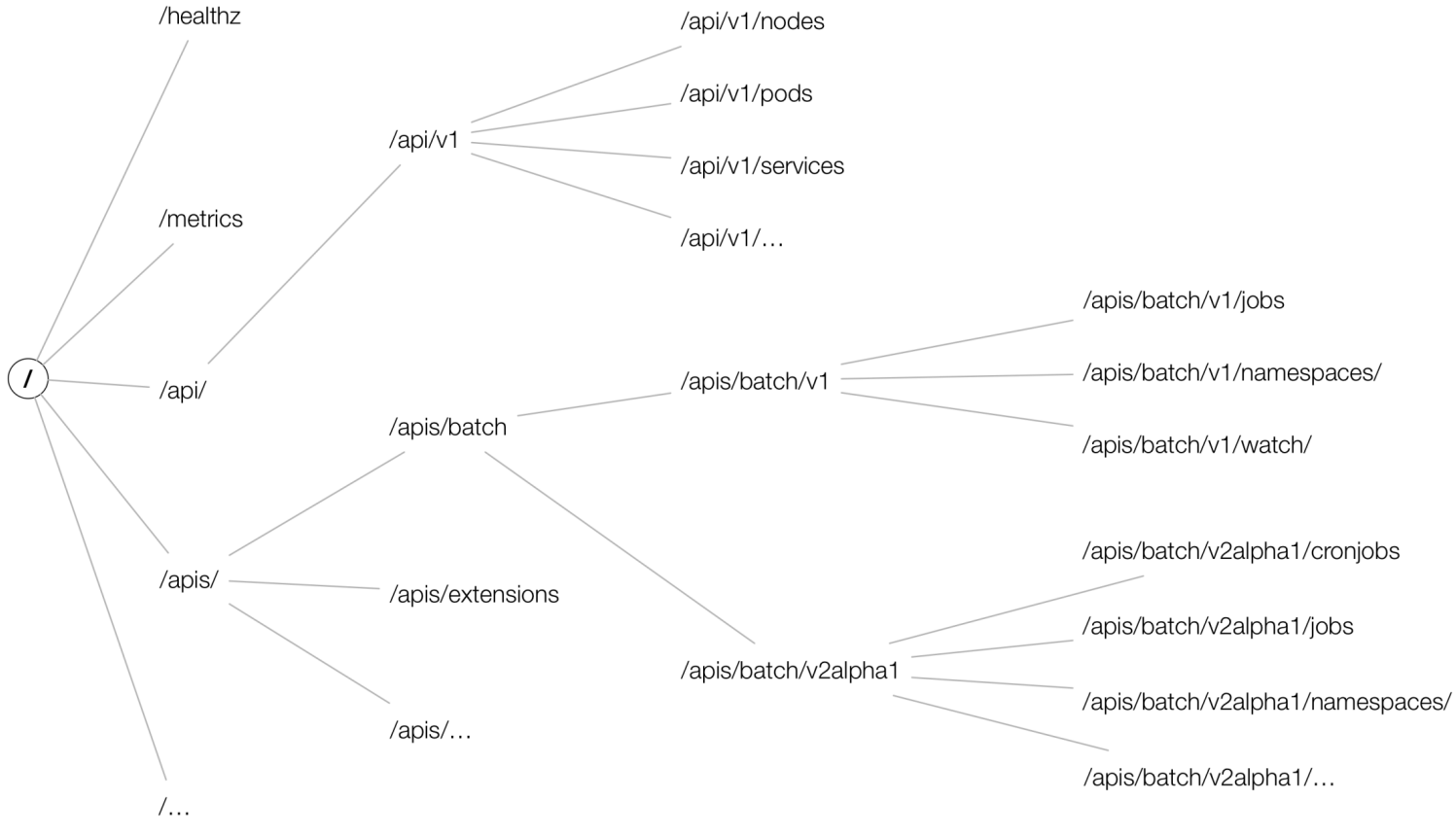
Группы API

Зачем?

- Разбиение монолитного v1 (core) API на группы и возможность включать/выключать их по отдельности
- Поддержка разных версий в разных группах (позволяет разработчикам разных API групп двигаться с разной скоростью)
- Возможность внедрения экспериментальных функций в experimental группе наравне с поддержкой стабильных версий объектов в их основных группах
- ...

Больше информации в design proposal

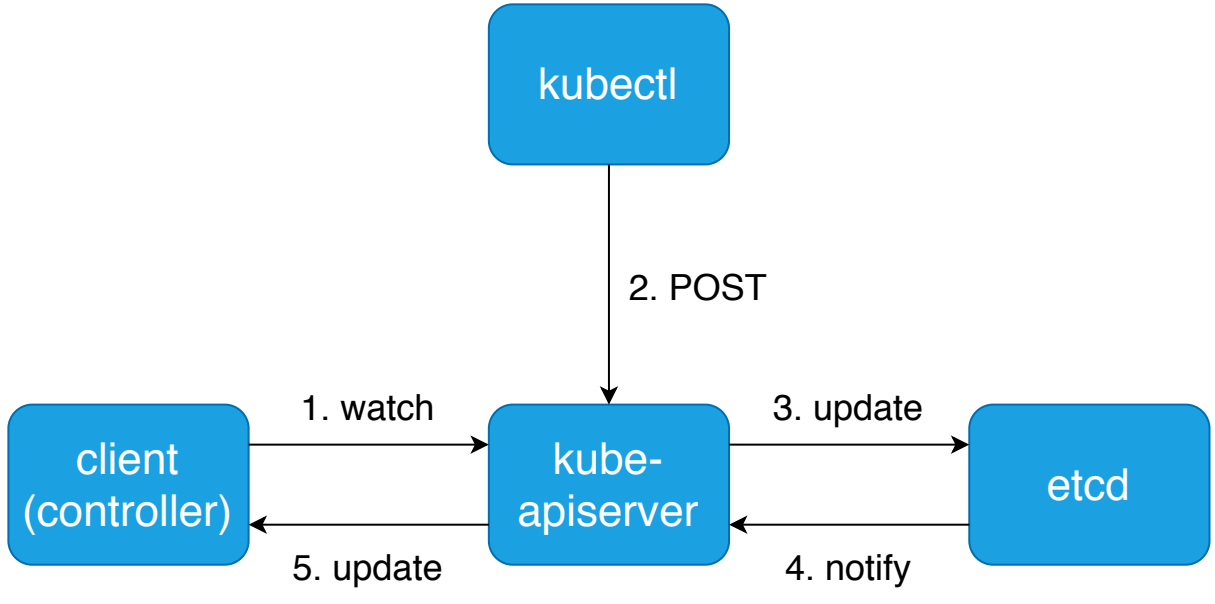
Группы API



Для исследования API очень удобно использовать postman

Модель подписок на изменения

- Клиенты подписываются (`watch`) на поток с определенными событиями (`events`)
- etcd при изменении объекта генерирует `event`



kubectl также может подписываться на события (`--watch`)

Как сломать API server

- Не следовать рекомендациям по размеру control-plane нод
- Напороться на баги или ошибиться в конфигурации других сервисов или компонентов (наиболее частая причина)

Демо. Подключаемся к kube-apiserver при помощи Postman

kube-controller-manager

Controller

Процесс отвечающий за приведение текущего состояния определенного ресурса к желаемому.

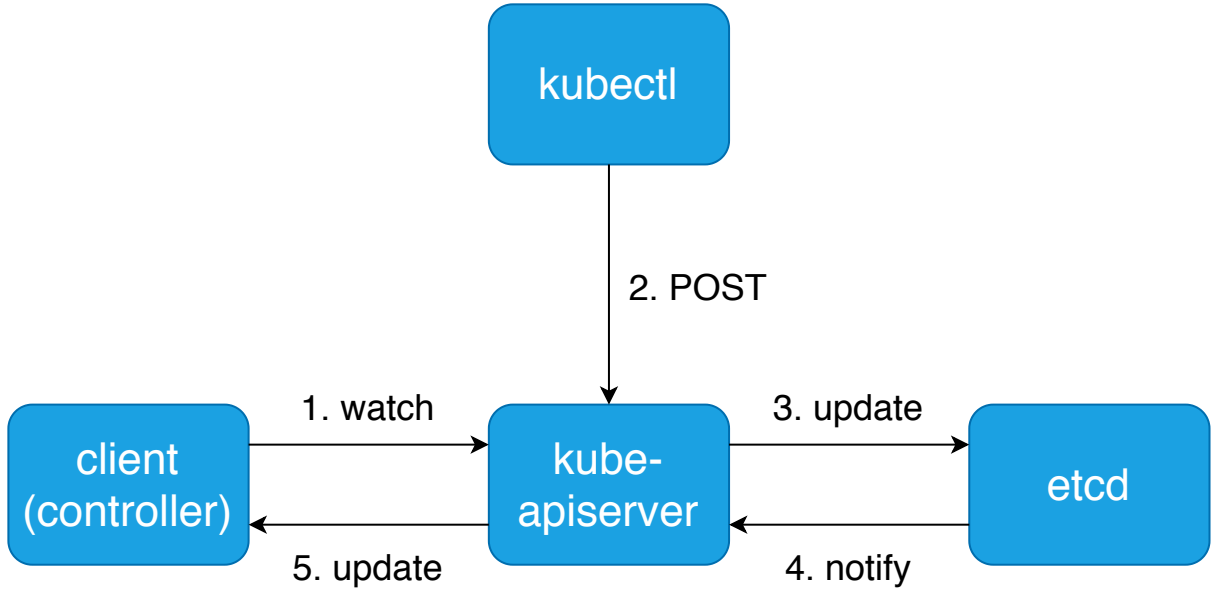
Желаемое состояние декларативно описывается в манифестах, либо генерируется динамически.

Упрощенное описание логики работы любого controller (reconciliation loop):

```
for {
  desired := getDesiredState()
  current := getCurrentState()
  makeChanges(desired, current)
}
```

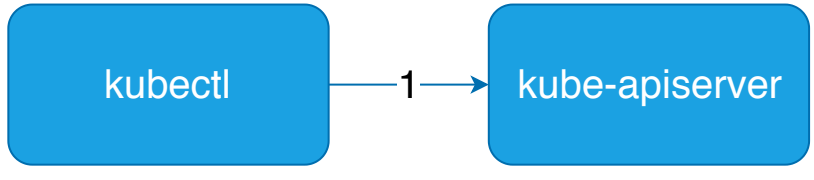
kube-controller-manager

- Набор контроллеров, встроенных в Kubernetes
- Каждый контроллер подписан на events об отслеживаемых объектах



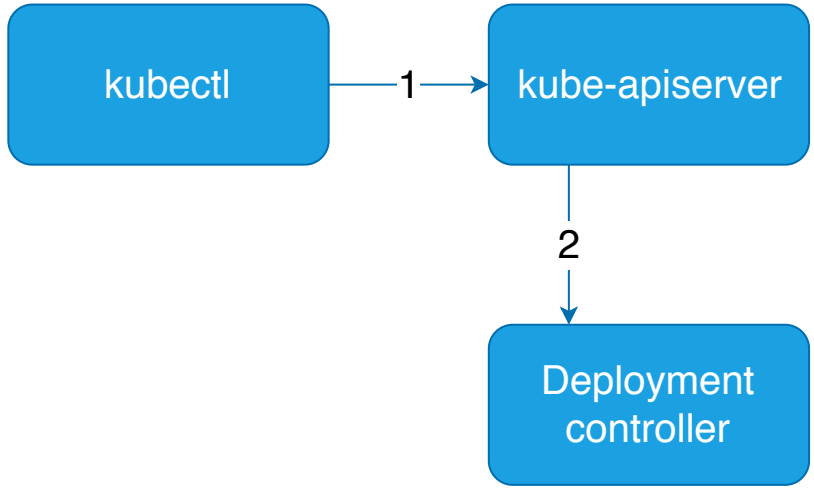
Создание Deployment

1. Через kubectl создаем `Deployment` объект (передаем `deploymentSpec` kube-apiserver)



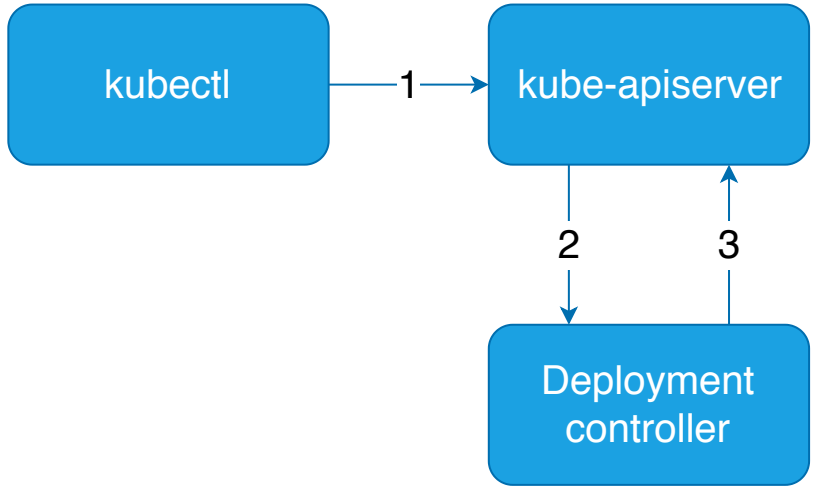
Создание Deployment

2. Deployment controller подписанный на event **Created** получает `deploymentSpec`



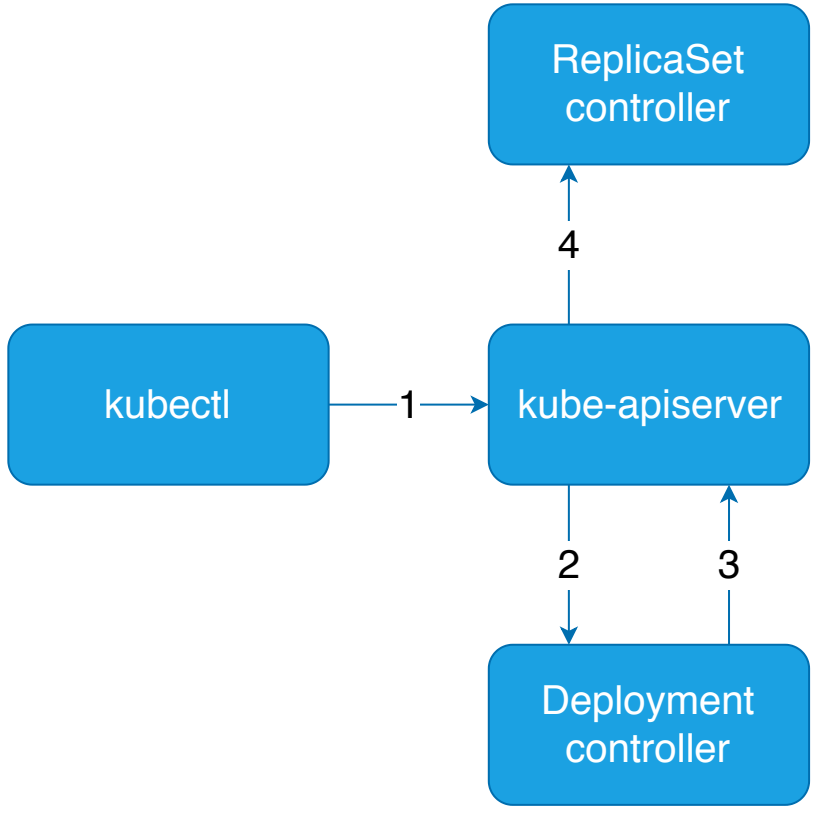
Создание Deployment

3. Deployment controller генерирует `replicaSetSpec` и возвращает его kube-apiserver



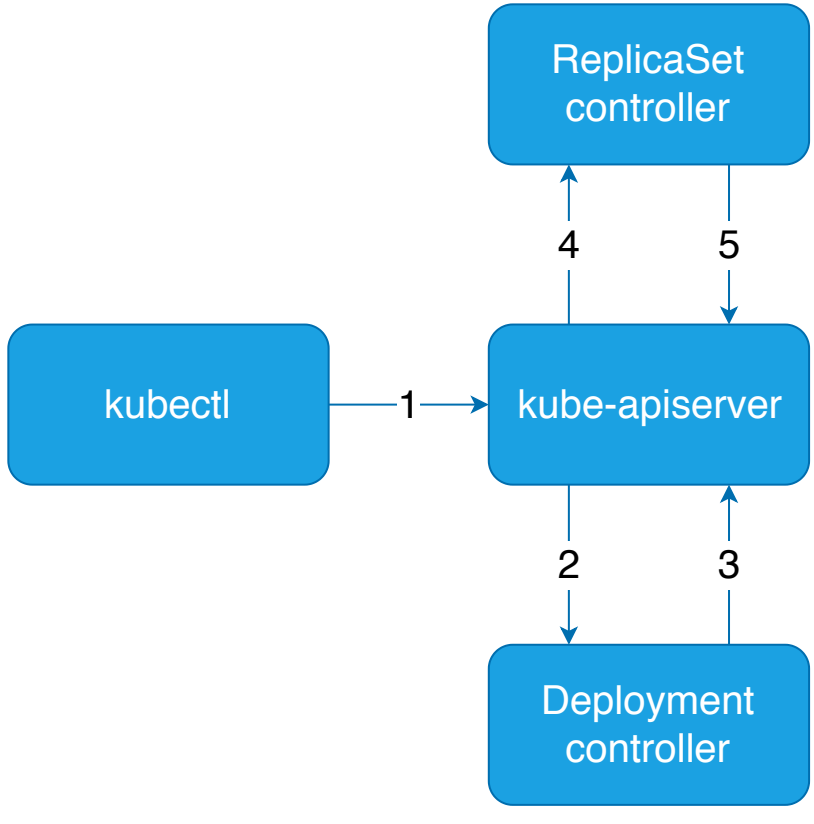
Создание Deployment

4. ReplicaSet conttroller подписанный на event **Created** получает `replicaSetSpec`



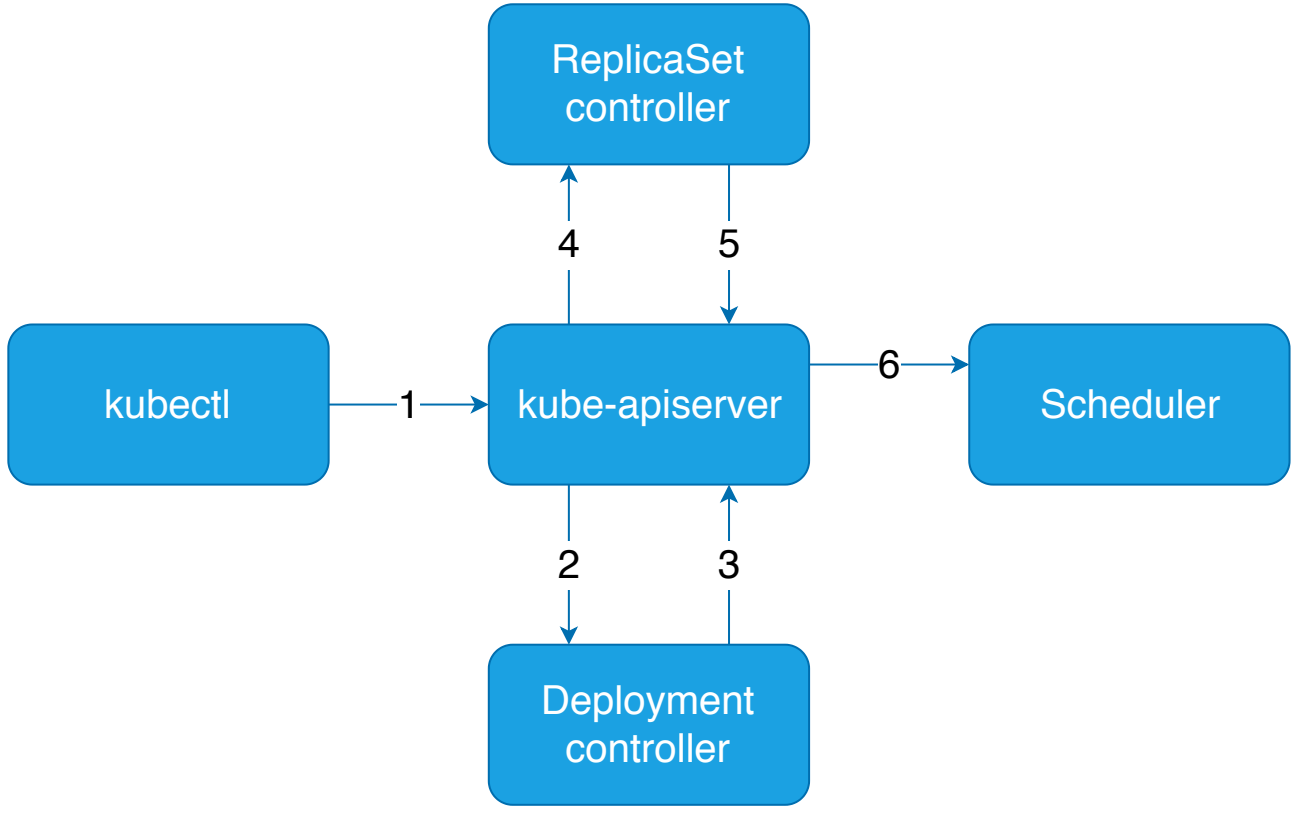
Создание Deployment

5. ReplicaSet controller генерирует podSpec и возвращает его kube-apiserver



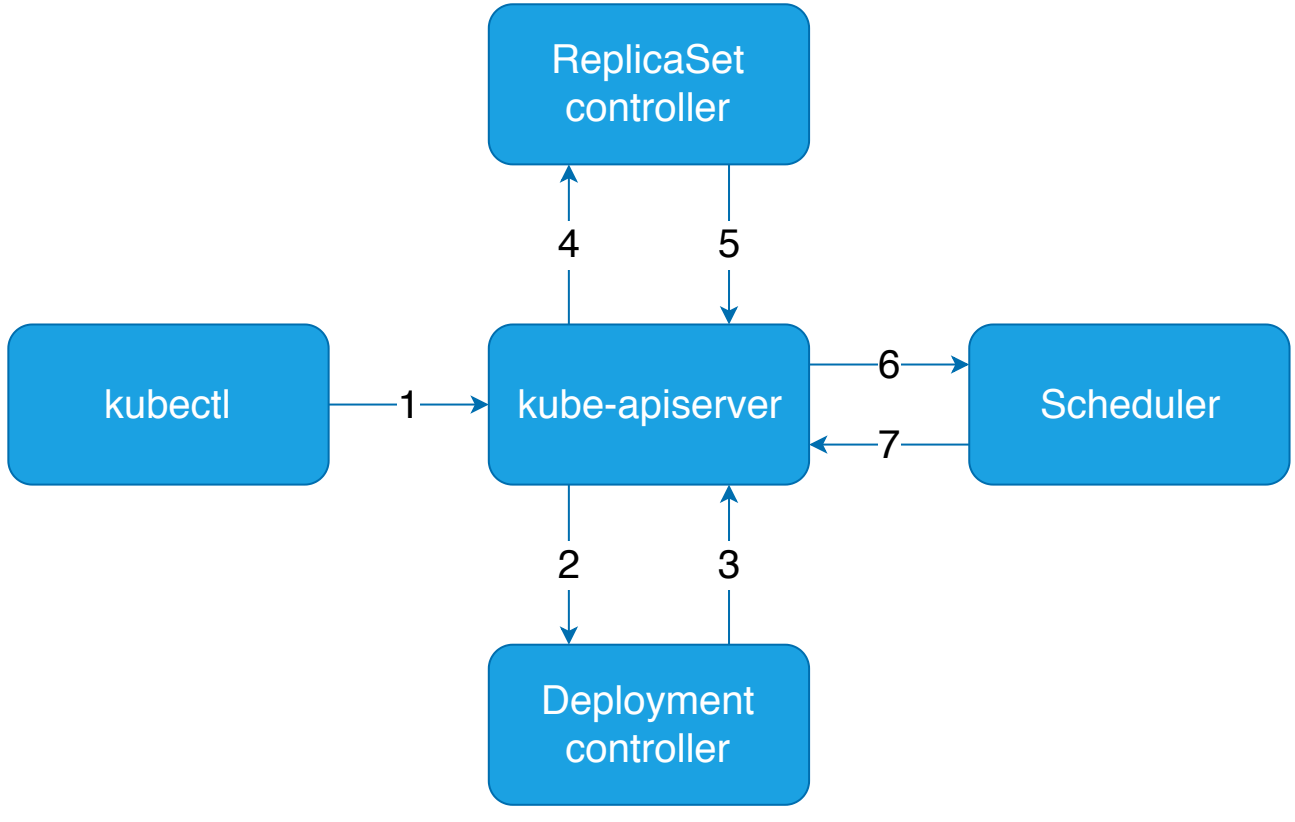
Создание Deployment

6. kube-scheduler подписанный на event **Created** получает podSpec



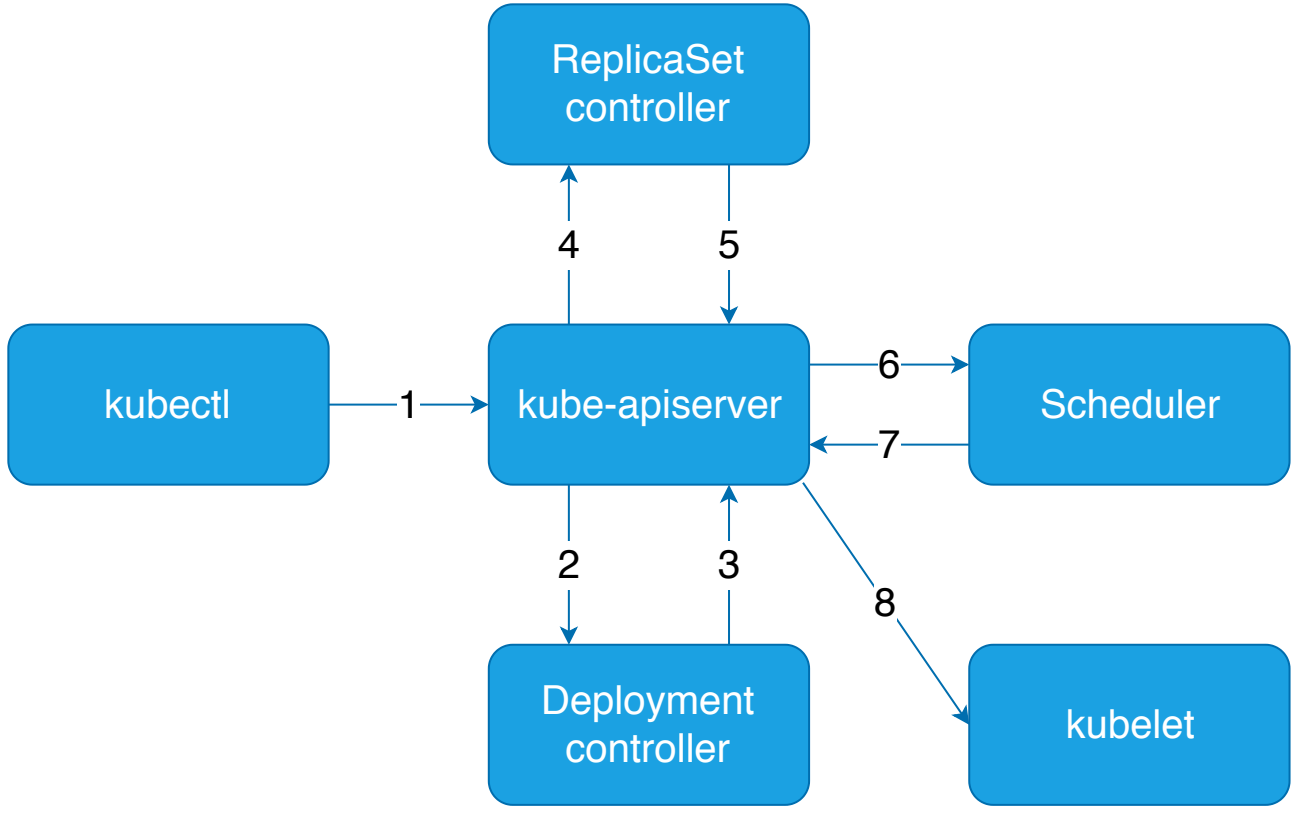
Создание Deployment

7. kube-scheduler делает **bind** pod на ноду и возвращает podSpec



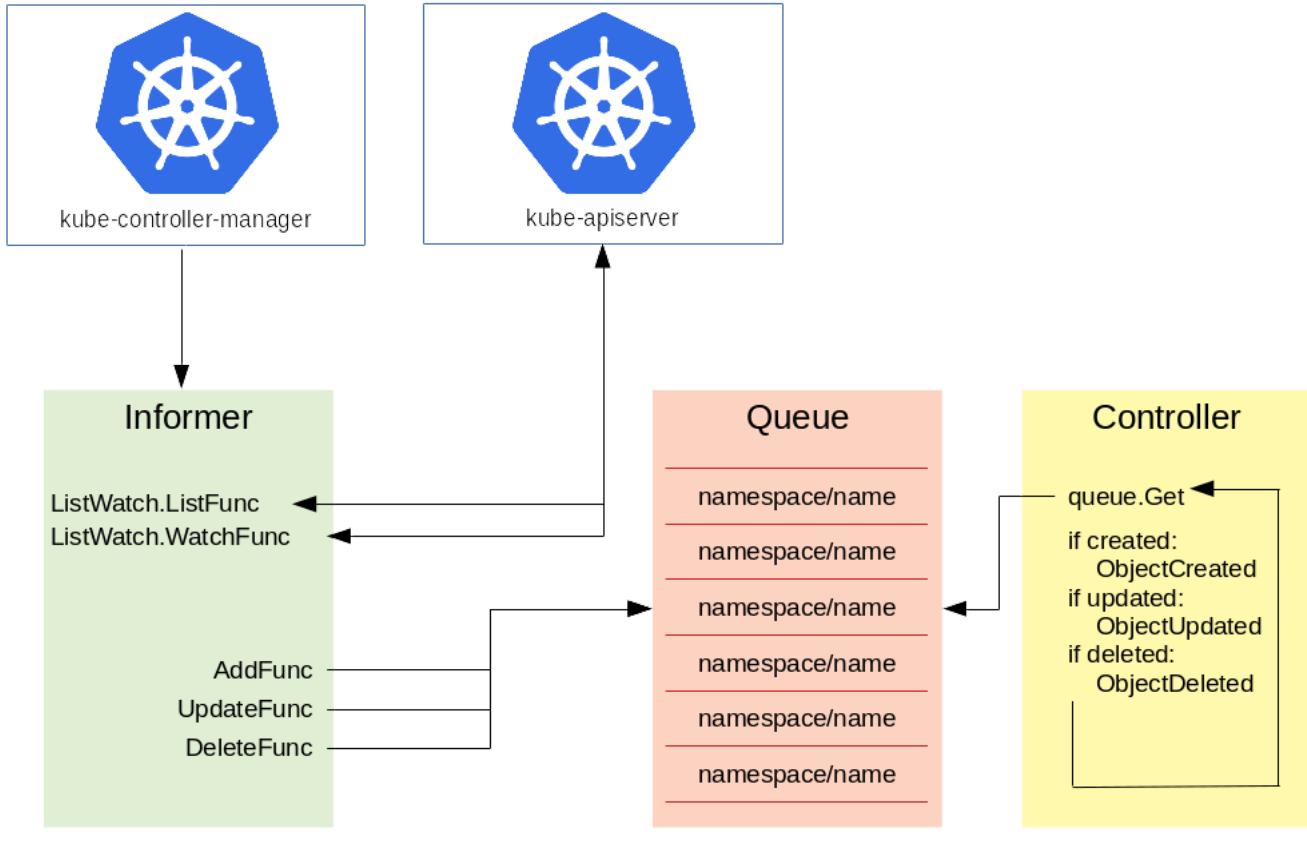
Создание Deployment

8. kubelet подписанный на event **Created** с указанием ноды получает podSpec и запускает pod



Controller

Логика обработки



Controller | компоненты

Informer/SharedInformer

- Отслеживает текущее и желаемое состояние объектов (`created`, `modified` и `deleted` events)
- Кэширует полученную информацию

> В отличии от обычного `Informer`, `SharedInformer` позволяет использовать единый кэш для разных контроллеров. Также в случае использования `SharedInformer` создается единая для всех контроллеров подписка на events.

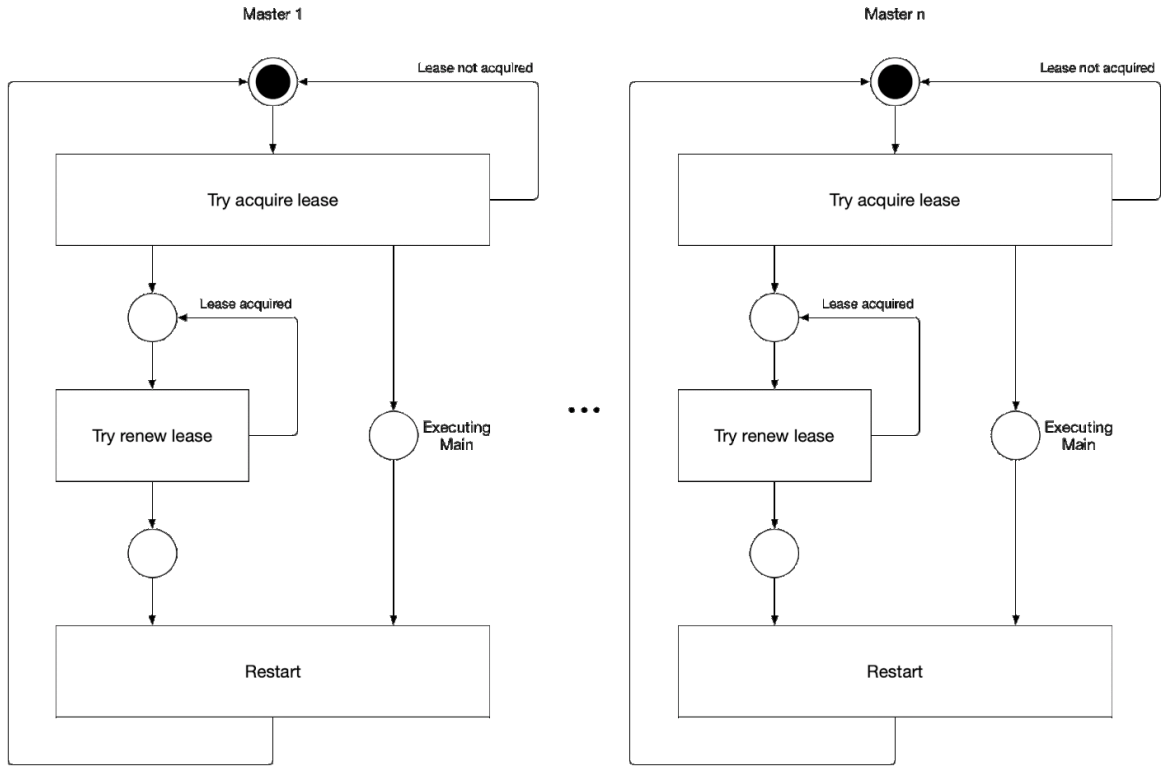
Resource Event Handler

Структура описывающая действия с объектами:

```
type ResourceEventHandlerFuncs struct {
    AddFunc    func(obj interface{})
    UpdateFunc func(oldObj, newObj interface{})
    DeleteFunc func(obj interface{})
}
```

- **AddFunc** - что должно произойти при добавлении объекта
- **UpdateFunc** - что должно произойти при изменении объекта
- **DeleteFunc** - что должно произойти при удалении объекта

Leader Election



Команда `kubectl describe endpoints kube-controller-manager -n kube-system` покажет какой controller-manager в данный момент является лидером

Несколько controller-manager в кластере

Что если выключить leader election для kube-controller-manager в HA кластере?

```
kubectl run nginx --image=nginx --replicas=1
```

```
Events:
LAST SEEN   TYPE      REASON                  OBJECT                          MESSAGE
62s         Normal    SuccessfulCreate        replicaset/nginx-7bb7cd8db5     Created pod: nginx-7bb7cd8db5-rcrgm
62s         Normal    SuccessfulCreate        replicaset/nginx-7bb7cd8db5     Created pod: nginx-7bb7cd8db5-hwk6z
62s         Normal    SuccessfulCreate        replicaset/nginx-7bb7cd8db5     Created pod: nginx-7bb7cd8db5-z5rpq
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-hwk6z
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-rcrgm
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-rcrgm
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-z5rpq
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-rcrgm
61s         Normal    SuccessfulDelete        replicaset/nginx-7bb7cd8db5     Deleted pod: nginx-7bb7cd8db5-hwk6z
60s         Normal    SuccessfulCreate        replicaset/nginx-7bb7cd8db5     Created pod: nginx-7bb7cd8db5-wpkdc
62s         Normal    ScalingReplicaSet       deployment/nginx                Scaled up replica set nginx-
7bb7cd8db5 to 1
```

kube-scheduler

kube-scheduler

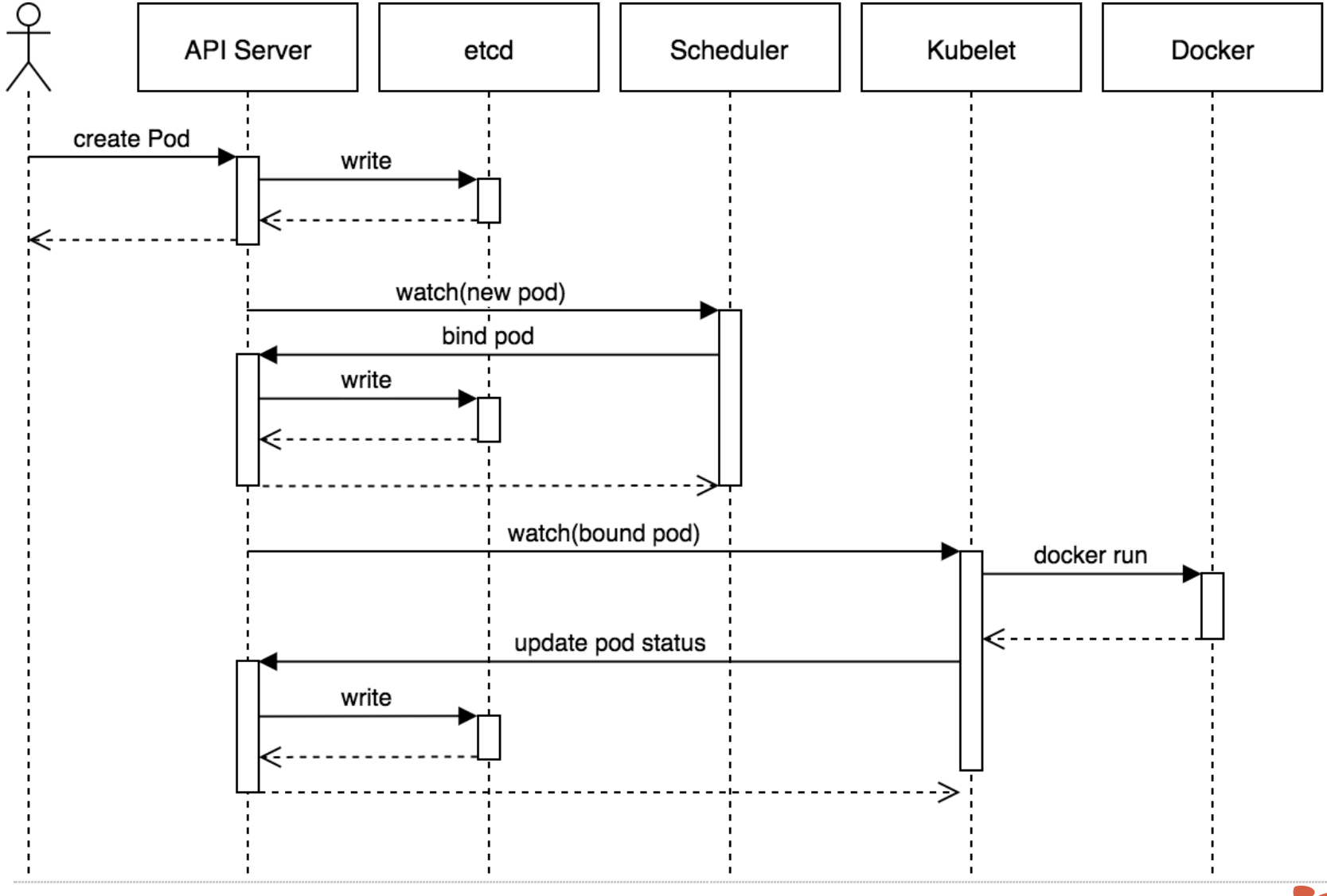
- Планировщик по умолчанию в Kubernetes
- Подписывается на события в kube-apiserver и управляет распределением pod на ноды

SIG Scheduling

- Отвечает за компоненты, принимающие решение о расположении pod
- Участвует в разработке **kube-scheduler** и аналогичных проектов, например descheduler, kube-batch или poseidon
- Поддерживает механизмы, позволяющие пользователю управлять распределением pod (например `Node Affinity`)

Описание SIG

Процесс запуска pod



Алгоритм работы kube-scheduler

- Помещение pod в очередь планирования
- Для каждого pod в очереди:
  - Фильтрация нод (computing predicates)
  - Приоритизация нод
  - Выбор ноды для запуска
  - Bind обрабатываемого pod на выбранную ноду

Общее верхнеуровневое описание

Predicates policies

Политики по фильтрации нод, на которых могут быть запущены pod. Проверка на соответствие данным политикам производится в определенном, заданном порядке.

Примеры:

- **CheckNodeConditionPred** - проверка, что нода:
  - Имеет статус `Ready`
  - Не находится в состоянии `NetworkUnavailable`
- **PodToleratesNodeTaintsPred** - проверка, что ноде не присвоены определенные taints, например `NoExecute` или `NoSchedule`

Больше примеров и исходный код на GitHub

Priority policies

Политики по приоретизации нод, на которых могут быть запущены pod.

Примеры:

- **LeastRequestedPriority** - алгоритм, расчитывающий приоритет (score) на основе суммарных `requests` pod, запущенных на ноде и ее общей мощности
- **ImageLocalityPriority** - алгоритм, расчитывающий приоритет на основе наличия на нодах образов контейнеров и их размера

Больше примеров и исходный код на GitHub

> Не все политики включены по умолчанию. Актуальный список для версии 1.15 можно посмотреть здесь

Недостатки

- Queue-based scheduling - может привести к неэффективной утилизации ресурсов
- Низкая скорость обработки запросов (пропускная способность)
- Статичное распределение pod (нет встроенных возможностей произвести rescheduling)

Планы на будущее

Poseidon-Firmament (Alpha, Release 0.8)

- **Firmament** - сторонний flow-based scheduler
- **Poseidon** - сервис для интеграции Firmament с Kubernetes
- Увеличивает производительность планирования в некоторых кейсах, результаты benchmark

> На текущий момент Poseidon-Firmament проигрывает kube-scheduler в случае, если у pod указаны правила affinity

Планы на будущее

Poseidon-Firmament (Alpha, Release 0.8)

Основные отличия от kube-scheduler:

- Bulk Scheduling
- Scheduling на основе реальной загрузки нод*
- Rescheduling
- Больше отличий

Планы на будущее

Scheduling Framework (Alpha в 1.15)

- Приведение планировщика к pluggable архитектуре
- Этапы работы планировщика разбиваются на части
- Каждая часть может быть реализована в виде подключаемых плагинов

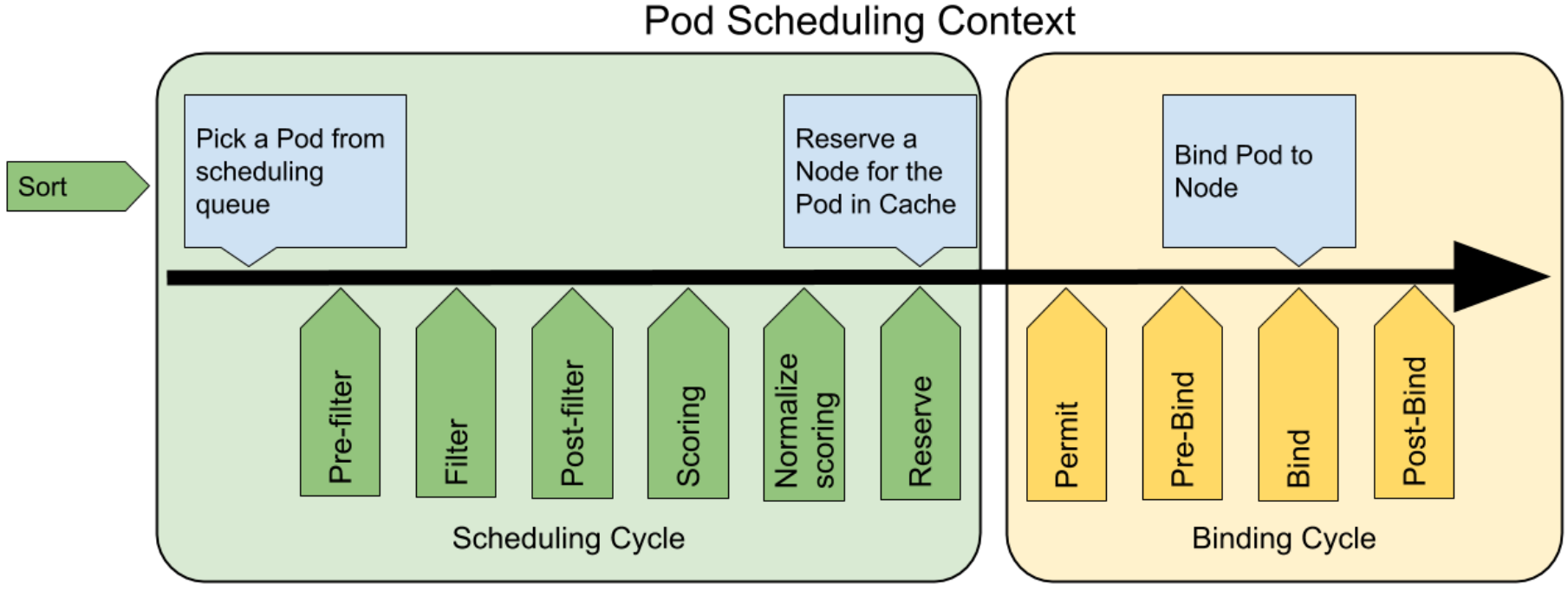
Две основных фазы:

- **Sheduling cycle**
- **Binding cycle**

KEP

Scheduling Framework

Каждая фаза разбивается на extension points (точки расширения):



Описание extension points

Несколько scheduler в кластере

1. Возможно явное указание собственного scheduler, который будет производить обработку запросов:

   podSpec:

```
schedulerName: custom-scheduler
```

2. Те scheduler, которые есть по умолчанию, выбирают лидера:

```
kubectl describe ep kube-scheduler -n kube-system

Events:
  Type    Reason          Age   From                Message
  ----    ------          ----  ----                -------
  Normal  LeaderElection  72m   default-scheduler   kind-control-plane_24798f04-0cdd-
45e7-8a8a-b128a814398e became leader
```

Несколько scheduler в кластере

Split Brain

Что если выключить leader election для default scheduler в HA кластере?

```
Events:
  Type     Reason            Age   From                 Message
  ----     ------            ----  ----                 -------
  Warning  FailedScheduling  3s    default-scheduler    Binding rejected: Operation cannot be fulfilled on pods/binding "nginx": pod nginx is already assigned to node "kind-worker3"
  Warning  FailedScheduling  3s    default-scheduler    Binding rejected: Operation cannot be fulfilled on pods/binding "nginx": pod nginx is already assigned to node "kind-worker3"
  Normal   Scheduled         3s    default-scheduler    Successfully assigned default/nginx to kind-worker3
```

Как жить со scheduler

Performance tuning

- **Percentage of Nodes to Score** - опция, позволяющая указать процент нод, который требуется найти на этапе фильтрации для начала приоретизации

  > Не будет работать, пока количество подходящих нод меньше 50, имеет смысл при больших размерах кластера

- **Predicates ordering** - расположение правил фильтрации нод в более подходящем для конкретного кластера порядке

Демо. Запускаем и используем свой scheduler

Управляем решением kube-scheduler

nodeSelector

Если не нужно сложных правил соотнесения pod/node - достаточно* использовать **nodeSelector**:

Присваиваем хосту метку:

```
$ kubectl label node fast-node kubernetes.io/disk=ssd
```

Указываем метку в спецификации:

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4 ...
5 spec:
6   nodeSelector:
7     kubernetes.io/disk: ssd
```

Минутка безопасности

Обычные метки хостов могут быть модифицированы kubelet.

Например, он управляет встроенными метками:

- `kubernetes.io/arch`
- `kubernetes.io/hostname`
- etc...

Используем **NodeRestriction** admission controller для создания гарантированно неизменяемых меток:

```
$ kubectl label node fast-node node-restriction.kubernetes.io/disk=ssd
```

Описание NodeRestriction

Affinity and anti-affinity

Более удобный способ управления решением Scheduler:

- Гибкий язык описания правил
- Возможность указывать `hard` и `soft` affinity
- Поддержка affinity на уровне pod (`inter-pod affinity`)

Операторы, используемые в правилах affinity:

- **`In`** & **`NotIn`**
- **`Exists`** & **`DoesNotExist`**
- **`Lt`** & **`Gt`**

Affinity and anti-affinity

Типы affinity:

- **requiredDuringSchedulingIgnoredDuringExecution** - хост должен удовлетворять условиям, чтобы scheduling прошел успешно (`Hard`)
- **preferredDuringSchedulingIgnoredDuringExecution** - хост должен удовлетворять условиям, чтобы выиграть scoring при прочих равных (`Soft`)
- **requiredDuringSchedulingRequiredDuringExecution** - хост должен удовлетворять условиям, чтобы scheduling прошел успешно и pod может быть вытеснен, если что-то изменилось (`Future`)

Node affinity

Если не удалось подобрать подходящий хост - не делаем `bind` :

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: hard-node-affinity
5  spec:
6    affinity:
7      nodeAffinity:
8        requiredDuringSchedulingIgnoredDuringExecution:
9          nodeSelectorTerms:
10         - matchExpressions:
11           - key: node-restriction.kubernetes.io/disk
12             operator: In
13             values:
14             - ssd
```

Node affinity and NodeSelector design proposal

Node affinity

Если не удалось подобрать подходящий хост - делаем `bind` на неподходящий:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: soft-node-affinity
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        preference:
          matchExpressions:
          - key: node-restriction.kubernetes.io/disk
            operator: In
            values:
            - ssd
```

Inter-pod affinity and anti-affinity

Правила размещения pod в соответствии с `topology domain` и расположением других pod.

Pod могут требовать запуск:

- В одном `topology domain`
- В разных `topology domain`

`Topology domain` определяются ключом **topologyKey**

> Inter-pod topological affinity and anti-affinity design proposal

Inter-pod anti-affinity

Расположить все pod в deployment на разных хостах:

```
apiVersion: apps/v1
kind: Deployment
...
spec:
...
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - nginx
            topologyKey: "kubernetes.io/hostname"
...
```

Inter-pod affinity

Расположить все pod в deployment на одном хосте:

```
apiVersion: apps/v1
kind: Deployment
...
spec:
...
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - nginx
            topologyKey: "kubernetes.io/hostname"
...
```

Taints (испорченность*)

Похожи на метки для хостов, но имеют служебное предназначение. Указываются в формате:

```
key=value:effect
complex/key="":effect
```

Эффекты `taints`:

- **NoSchedule** - не располагать pod на хосте
- **PreferNoSchedule** - не располагать pod на хосте (но если очень надо, то можно)
- **NoExecute** - вытеснять pod с хоста (в общем случае)

Например, `taint` на control-plane хостах:

```
node-role.kubernetes.io/master:NoSchedule
```

Taints

Taints, говорящие о проблемах с хостом:

- **node.kubernetes.io/not-ready**
- **node.kubernetes.io/out-of-disk**
- **node.kubernetes.io/memory-pressure**
- ...

На основании данных taints NodeController может выставлять хосту taint `NoSchedule`, который учитывается kube-scheduler (**TaintNodesByCondition**).

Возможно, документация не совсем корректна в описании того, учитывает ли kube-scheduler node conditions ?

Tolerations (терпимость*)

Указание pod запускаться на хостах, помеченных определенными taints:

```
1  apiVersion: extensions/v1beta1
2  kind: DaemonSet
3  metadata:
4    name: toleration-ds
5  spec:
6    template:
7      spec:
8        tolerations:
9        - effect: NoExecute
10         operator: Exists
11       - effect: NoSchedule
12         operator: Exists
```

Демо. Управляем решением kube-scheduler

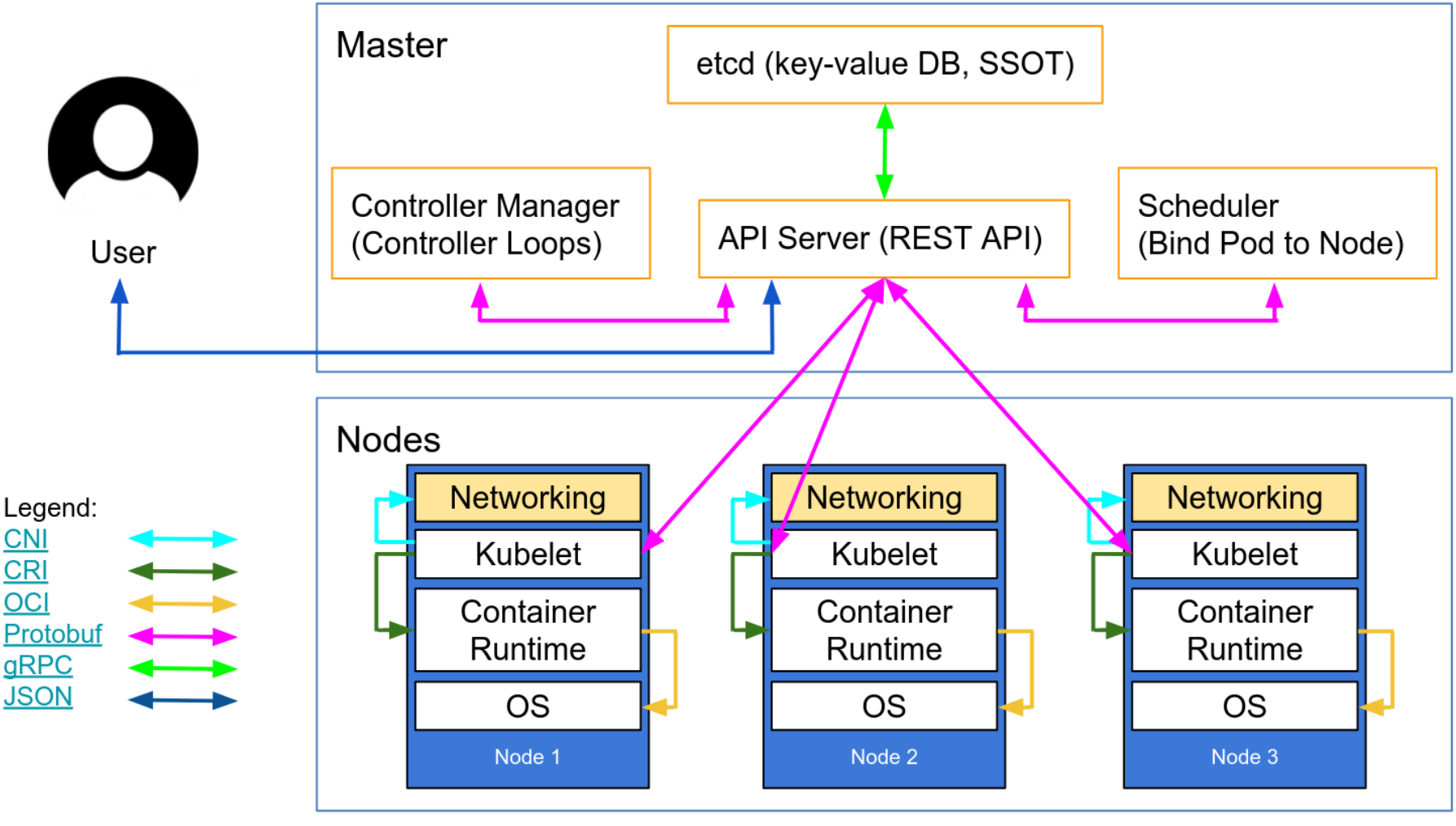
kubelet

kubelet | взаимодействие

Используется несколько протоколов и интерфейсов. В их числе:

- **CRI (Container Runtime Interface)**:
  - Коммуникация с container engine (runtime)
  - Стандартный интерфейс, позволяющий подключать разные runtime
  - gRPC + protobuf
- **CNI (Container Network Interface)**:
  - Коммуникация с сетевыми плагинами
  - Сетевые плагины вызываются kubelet и предоставляют IPAM, а также настраивают сетевые интерфейсы для pod

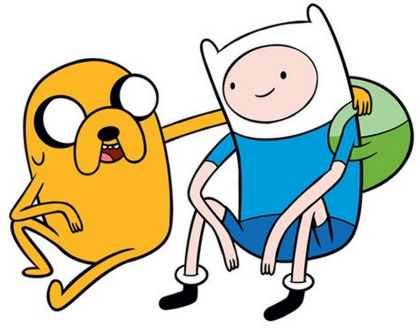
kubelet | взаимодействие



Static pods

- Pod'ы, которые управляются kubelet и не привязаны к kube-apiserver/kube-scheduler/kube-controller-manager
- Как правило, системные компоненты, запуск которых нужен до запуска control-plane (сам control-plane)
- Не рекомендуется использовать для целей, которые могут закрыть DaemonSets

```
--pod-manifest-path=</etc/kubernetes/manifests>
--manifest-url=<URL>
```



Спасибо за внимание!

Время для ваших вопросов!