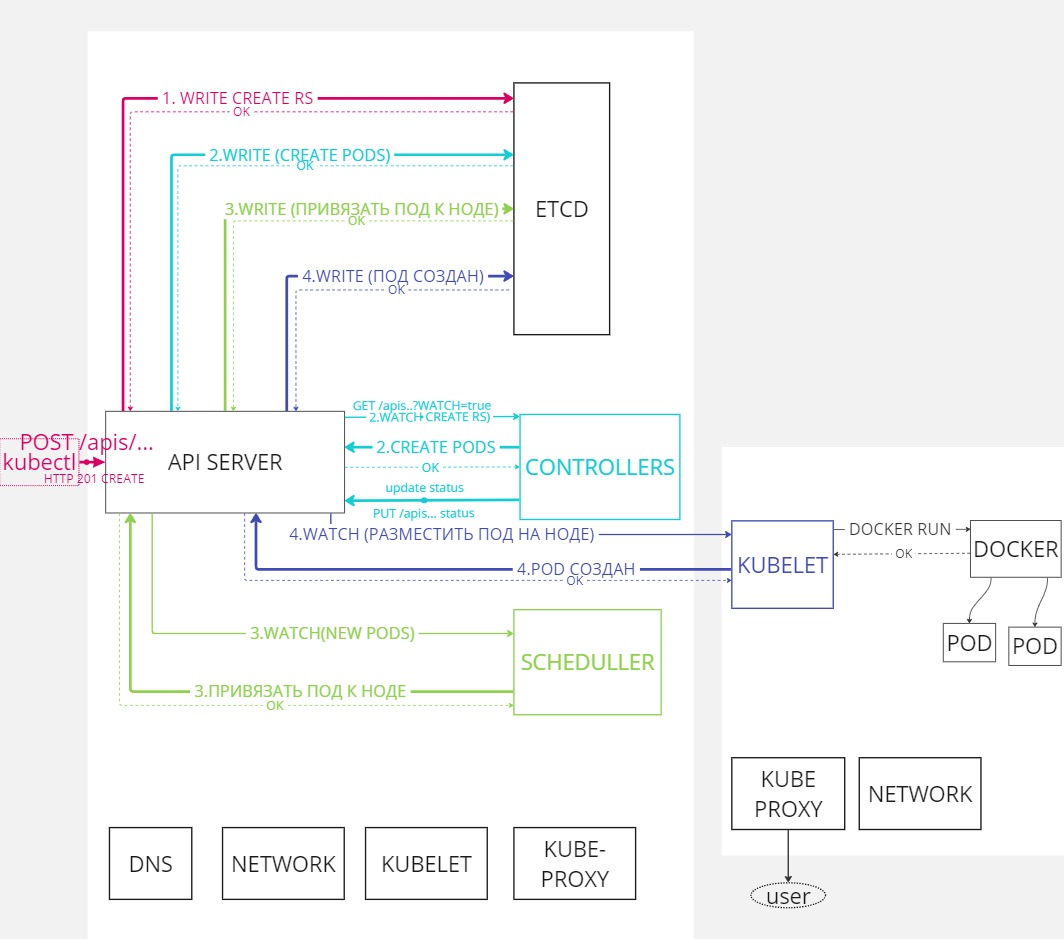
**Архитектура**



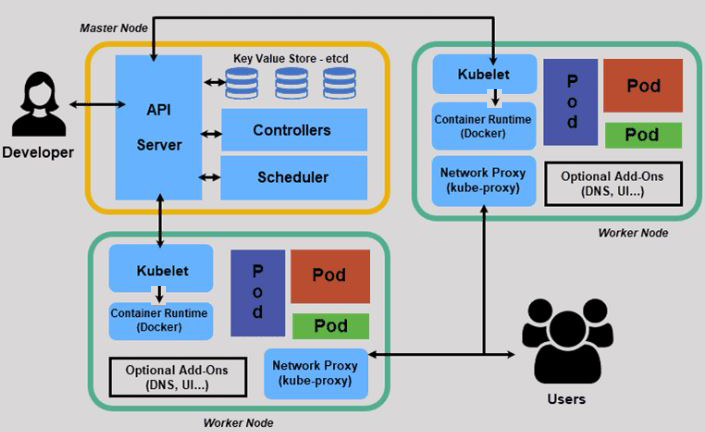
Цепь событий:

1.kubectl отправляет манифест в API (HTTP POST). API проверяет и записфвает в ETCD и возращает ответ kubectl

2.контроллер через механизм наблюдения получает уведомление о создании deploy, создает RS через API

3.информация о созданных подах хранится в etcd, Планировщик видя такой под назначает ему лучшую ноду

4.kubelet наблюдая за изменениями в подах на API видит новый назначенный ему под и через docker запускает конт-ры пода



**Элементы плоскости управления:**

**Etcd** Распределённое и высоконадёжное хранилище данных в формате "ключ-значение", которое используется как основное хранилище всех данных кластера в Kubernetes

**API server** Общение с кластером kubernetes происходит через API server (интерфейс взаимодействия между программами)

Например манифест файл (создание подов/деплоймента и тд) передаем в кластер через API сервер. Kubectl через rest api общается с кластером. Он же ограничивает доступ к rest api, таким образом ограничиваем доступ к каким-то компонентам k8s

**Scheduler** Компонент, который отвечает за размещение подов на конкретных нодах кластера, исходя из множества факторов - требования к ресурсам, affinity (anti-affininty) местонахождения данных и тд

**Controller-manager** Отвечает за то, что вы описали в etcd появилось в кластере

Компонент control plane запускает процессы *контроллера* (управляющий цикл который отслеживает общее состяние кластера через API-сервер и вносит изменения пытаясь привести текущее состояние к желаемому состоянию)

Эти контроллеры включают:

- Контроллер узла (Node Controller): уведомляет и реагирует на сбои узла.

- Контроллер репликации (Replication Controller): поддерживает правильное количество подов для каждого объекта контроллера репликации в системе.

- Контроллер конечных точек (Endpoints Controller): заполняет объект конечных точек (Endpoints), то есть связывает сервисы (Services) и поды (Pods).

- Контроллеры учетных записей и токенов (Account & Token Controllers): создают стандартные учетные записи и токены доступа API для новых пространств имен

**Компоненты работающие на рабочих узлах:**

**Kubelet** непрерывно отслеживает API, взаимодействует с системой контейнеризации (**docker/containerD/crio**) которая запускает и удаляет контейнеры пода. Отслеживает поды и сообщает их статус в API

Делает liveness rediness startup пробы (перезапускает когда не проходят), *стоит на всех нодах*

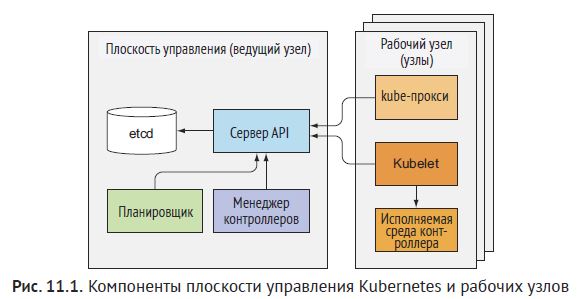
**Kube-proxy** Организация работы сервисов (iptables) *стоит на всех нодах*

**Дополнительные компоненты:**

DNS сервер

Сетевой драйвер

Ingress контроллер



Компоненты k8s взаимодействуют только с сервером **API** , **API** единственный компонент взаимодействующий с **etcd**, все остальные компоненты читают и записывают данные в хранилище **etcd** косвенно через сервер **API**

**ETCD:**

Хранилище, которое представляет собой быстрое распределенное и согласованное хранилище в формате ключ-значение

*Просмотр данных в базе данных:*

Добавить переменные сертификатов:

export ETCDCTL\_CACERT=/var/lib/certs/etcd/ca.crt \

export ETCDCTL\_CERT=/var/lib/certs/etcd/peer.crt \

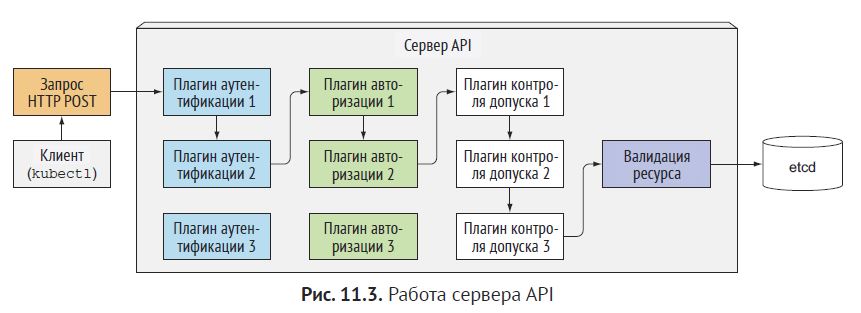
export ETCDCTL\_KEY=/var/lib/certs/etcd/peer.key \

export ETCDCTL\_API=3

команда:

etcdctl get --keys-only --prefix=true /registry/pods/

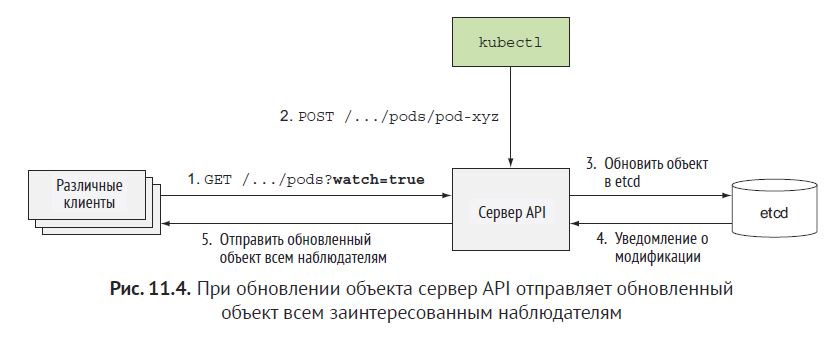
**API server:**



*Аутентификация* - плагин извлекает логин, идентификатор и группу принадлежащие пользователю, эти данные используются для авторизации.

*Авторизация* - плагин проверяет может ли пользователь выполнять запрошенное действие на ресурсом, например может ли пользователь создавать поды в запрошенном пространстве имен

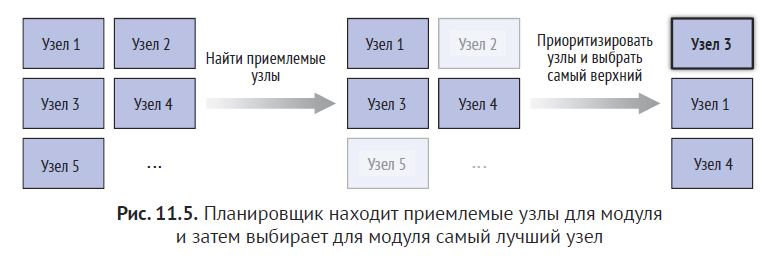
*Как сервер API уведомляет об изменениях ресурсов:*



**Scheduller:**

*Алгоритм назначначения подов на ноды:*

Фильтрация нодов на которые можно поставить под и приоретизация нодов и выбор лучшего, если несколько нодов имеют самый высокий балл, используется циклический перебор, чтобы гарантировать, что модули развернуты равномерно по нодам



*Критерии выбора ноды:*

- может ли под выполнять запрос аппаратных ресурсов

- заканчиваются ли в ноде ресурсы

- запрос пода на конкретную ноду (например по имени)

- запрос пода на порт хоста

- запрос пода на тип тома, может ли этот том быть смонтирован для этого пода на этой ноде

- допускает ли под ограничения узла

- сходство / антисходство ( affinity antiAffinity)

**CONTROLLERS:**

*Список контроллеров:*

- ReplicationController

- ReplicaSet

- Deployment

- StatefulSet

- контроллер узла

- контроллер службы Service

- контроллер конечных точек Endpoints

- контроллер пространства имен Namespaces

- контроллер постоянного тома PersistentVolume

- другие

*Исходный код контроллеров:*

<https://github.com/kubernetes/kubernetes/tree/master/pkg/controller>

***RC, ReplicaSet, Deploy, DaemonSet, Job***

Работают походим образом, следят за требуемым кол-вом подов и фактическим, при необходимости добавляют/удаляют, приводя к состоянию из манифеста

***StatefulSet***

Работает подобно RS, но в отличии от других контроллеров, которые только лишь управляют подами, он также создает поды и управляет заявками PersistentVolumeClaim для каждого пода

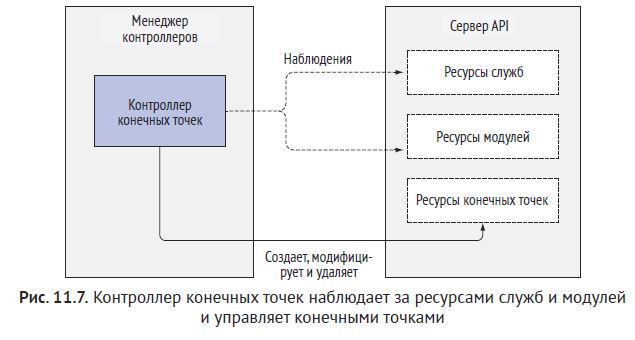
***Котроллер узла***

Управляет ресурсами ноды, отслеживает работоспособность каждой ноды и удаляет поды из недоступных нод

***Контроллер конечных точек***

Постоянно обновляет список конечных точек IP-адресами и портами модулей, соответсвующих селектору меток

Также удаляет объект конечных точек при удалении службы



***Контроллер пространства имён***

При удалении ресурса пространства имен Namespaces все ресурсы в этом пространстве имен должны быть удалены, это делает контроллер NS

***Контроллер постоянного тома***

Привязывает PV к PVC. Когда возникает PVC контроллер выбирает наименьший PV с режимом доступа в заявке и объявленной ёмкости выше ёмкости в заявке. Затем, когда удаляется заявка PVC, том отсоединяется и освобождается в соответвии с политикой (остается как есть, удаляется, очишается)