
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Меня хорошо видно && слышно?

Ставьте  , если все хорошо

Напишите в чат, если есть проблемы

Проверить, идет ли запись!

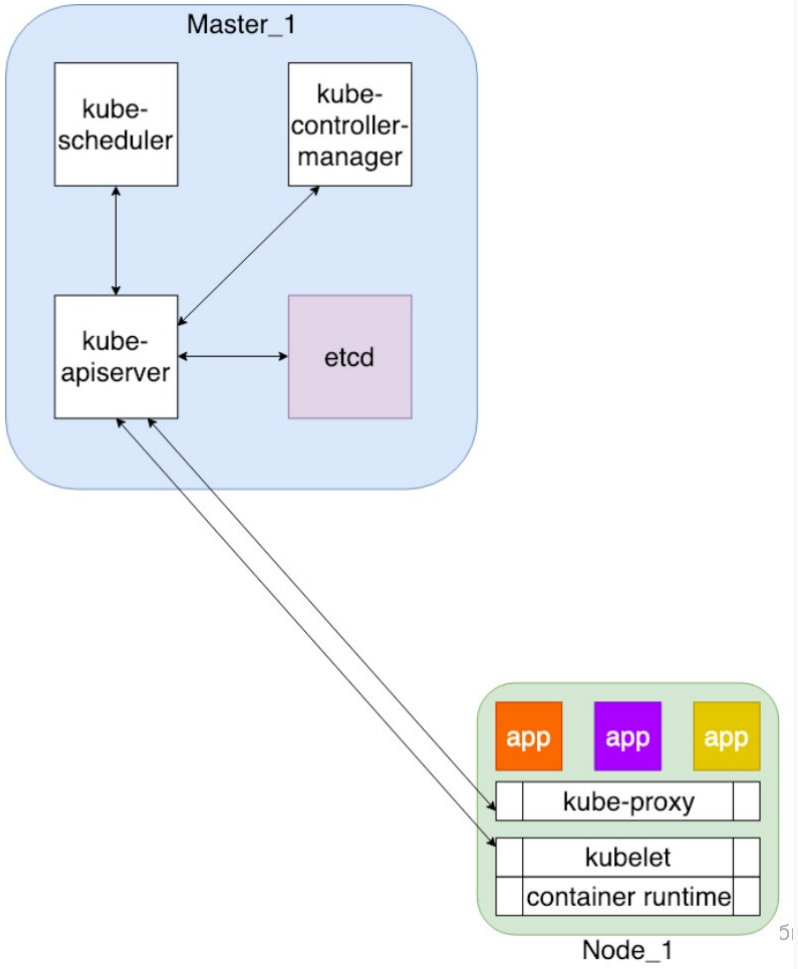
Архитектура Kubernetes

План

1. Архитектурные компоненты Kubernetes
2. Основные концепции Kubernetes

Начнем

Архитектура Kubernetes



Node – узел, который может быть виртуальной или физической машиной, в зависимости от кластера. Каждая node содержит службы, необходимые для запуска pods, управляемых control plane.

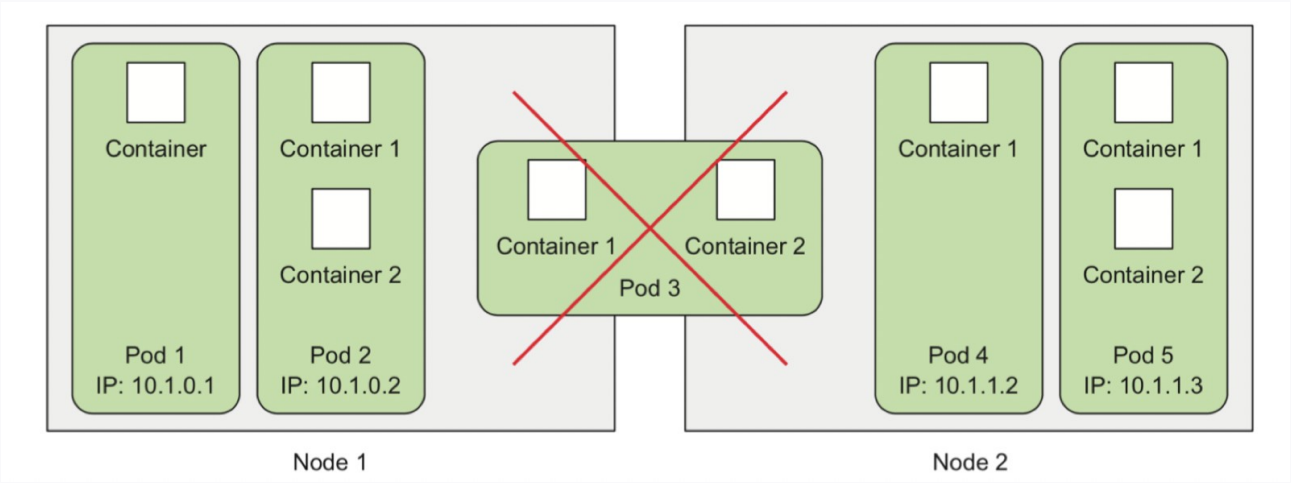
Master node – компоненты уровня управления

Worker node – размещает pods, что являются компонентами рабочей нагрузки

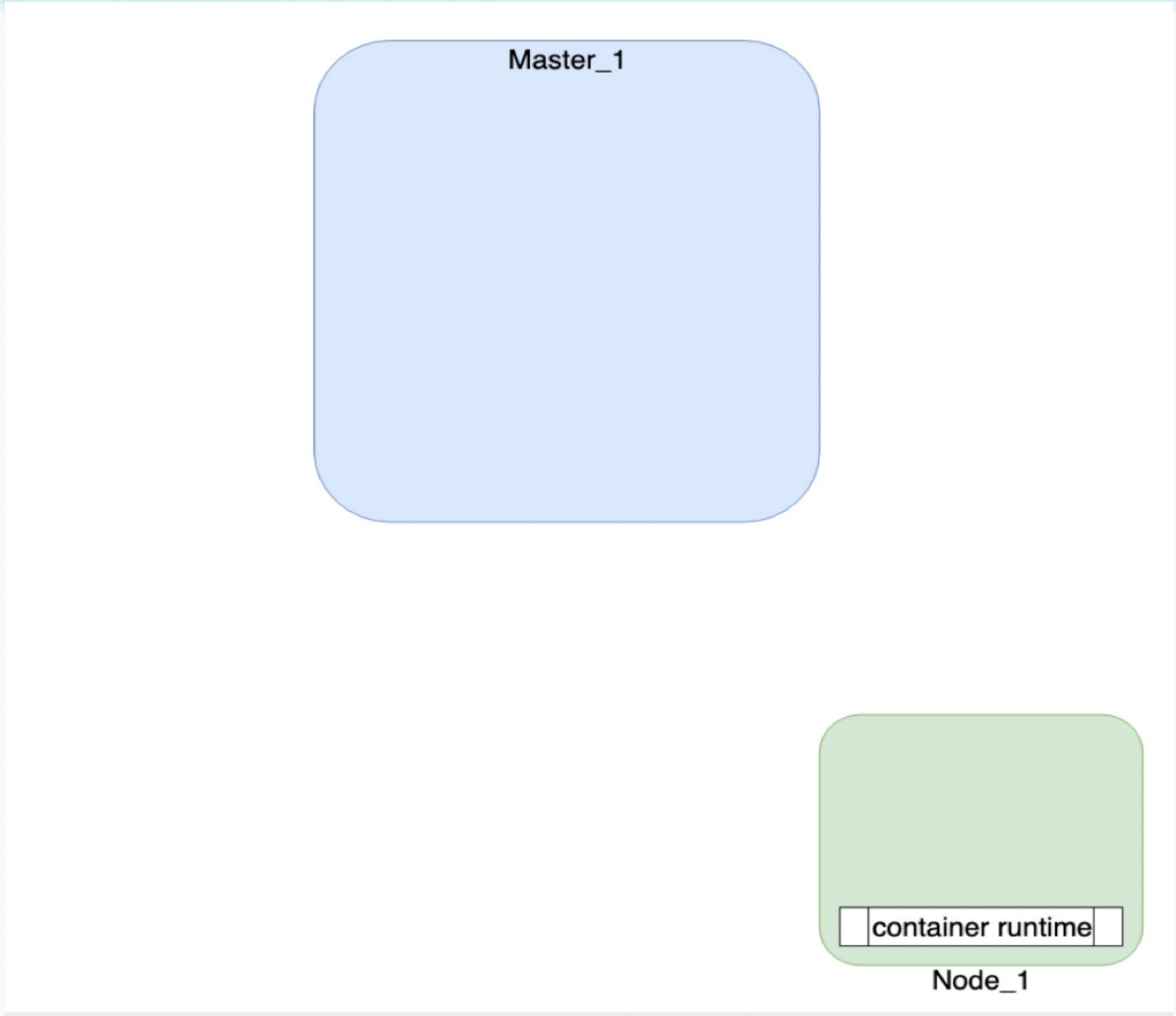
PODs

POD это:

- Минимальная сущность, управляемая Kubernetes
- Группа из одного или более контейнера с общим хранилищем/сетевыми ресурсами и спецификацией как запускать контейнеры
- Отдельный инстанс приложения

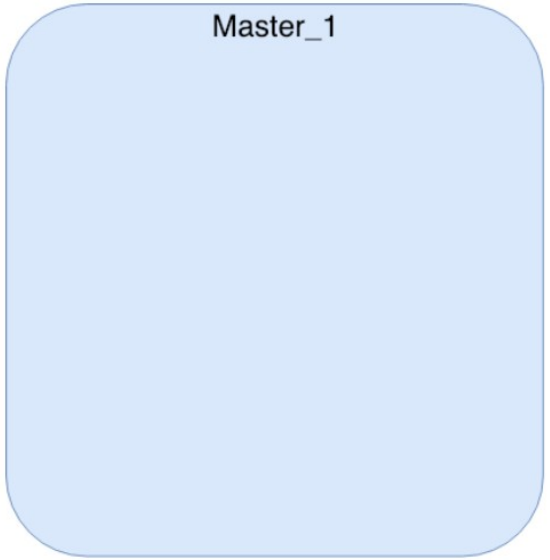


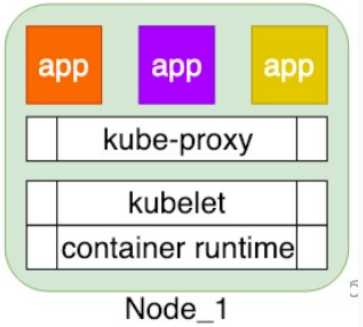
Container runtime



Docker
Container D
Rocket

Worker node: kubelet и kube-proxy





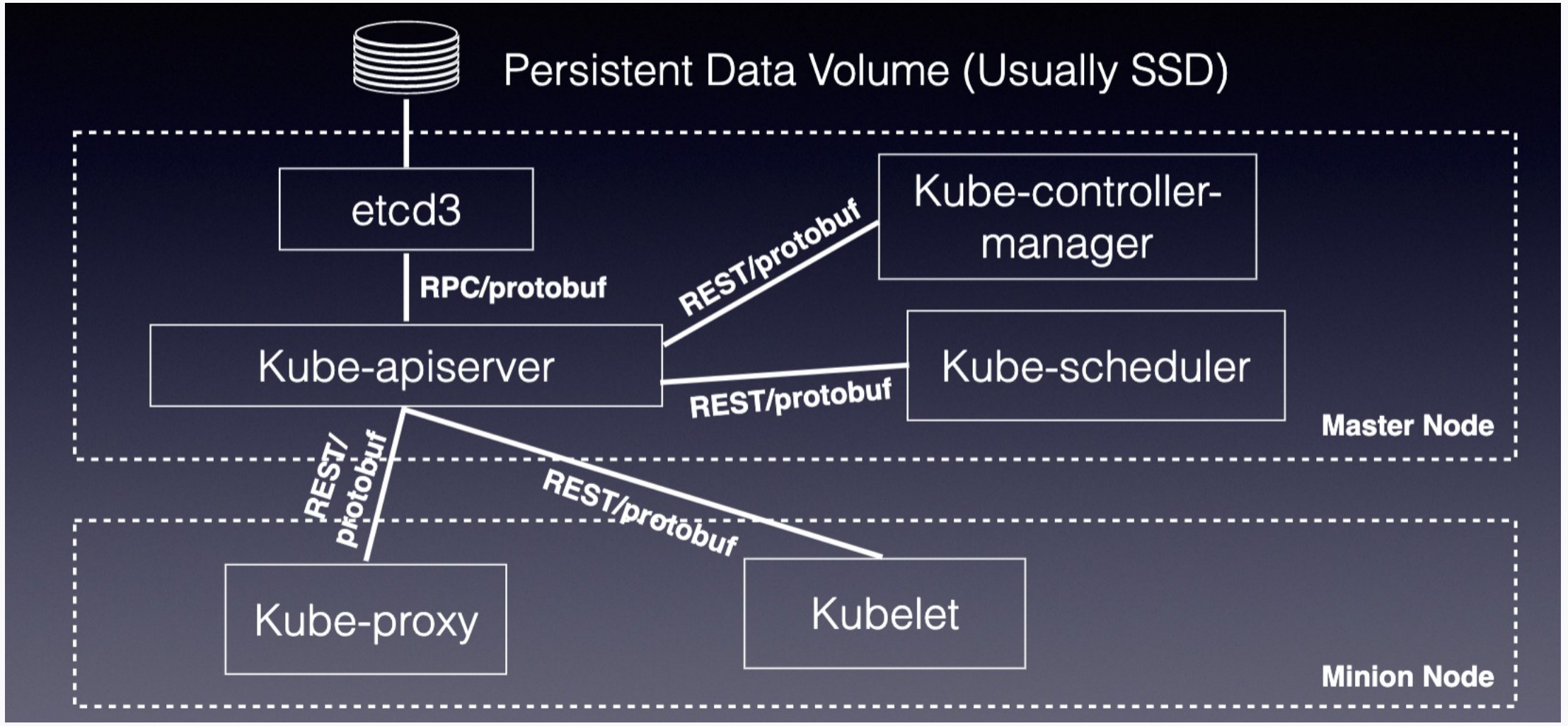
kubelet

Агент, который работает на каждом узле кластера.
Гарантирует, что контейнеры работают в Pod.
Принимает набор PodSpecs, которые предоставляются через различные механизмы, и гарантирует, что контейнеры, описанные в этих PodSpecs, работают и исправны.
Не управляет контейнерами, которые не были созданы Kubernetes

kube-proxy

это сетевой прокси, который запускается на каждом узле вашего кластера и реализует часть концепции Kubernetes Service.
Поддерживает сетевые правила на узлах.
Эти сетевые правила разрешают сетевую связь с Pods из сетевых сеансов внутри или вне вашего кластера.

Kubernetes Control Plane



Kube-apiserver & Etcd

Основная функциональность:

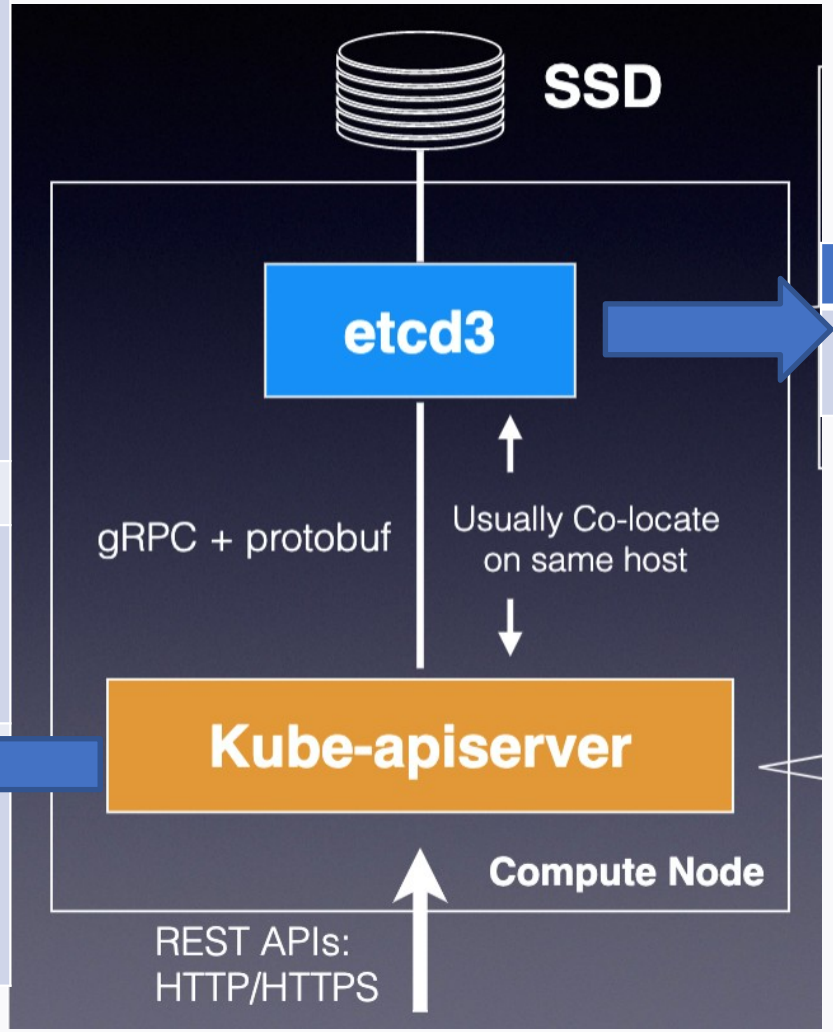
- Аутефикация
- Контроль допуска (RBAC)
- Ограничение скорости
- Объединение пулов
- Группировка функций
- Подключаемое внутреннее хранилище
- Определение подключаемого клиентского API
- Отслеживание клиентов и статистика вызовов API

Для объектов API:

- Операции CRUD
- Проверка
- Контроль версий
- Кэширование

Для контейнеров:

- Проксирование приложения
- Журналы контейнеров потока
- Выполнить команду в контейнере
- Метрики



Etcd как хранилище метаданных

- Наблюдаемое распределенное хранилище K-V

Put it together

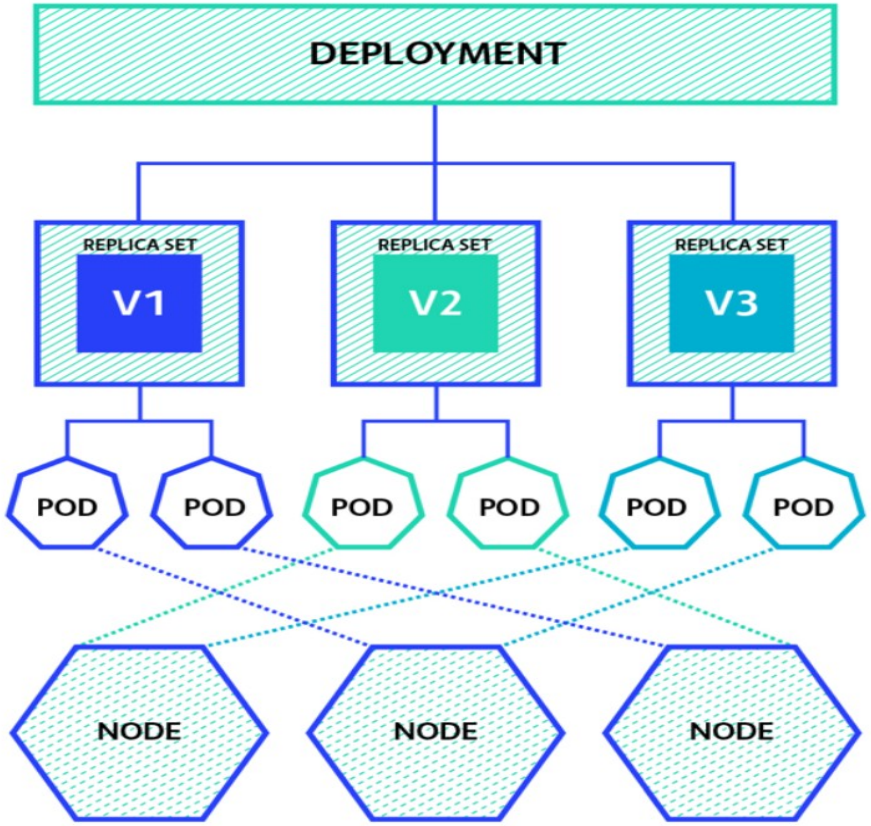
Что произойдет если выполнить следующую команду

```
$ kubectl create -f myapp.yaml
Deployment nginx-deployment created
$
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
     containers:
     - name: nginx
       image: nginx:1.7.9
       ports:
       - containerPort: 9376
```

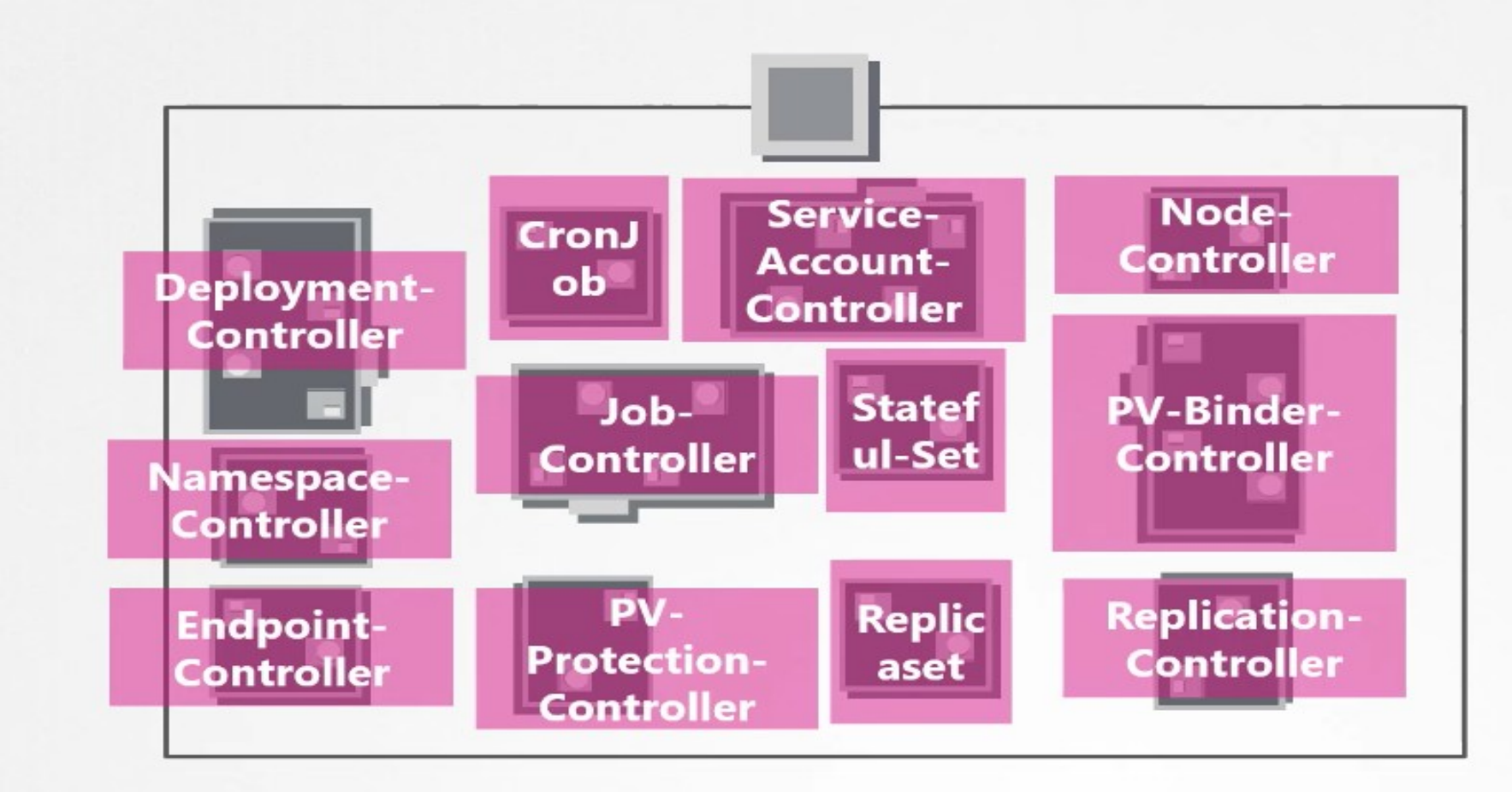
Основные концепции Kubernetes

Deployment

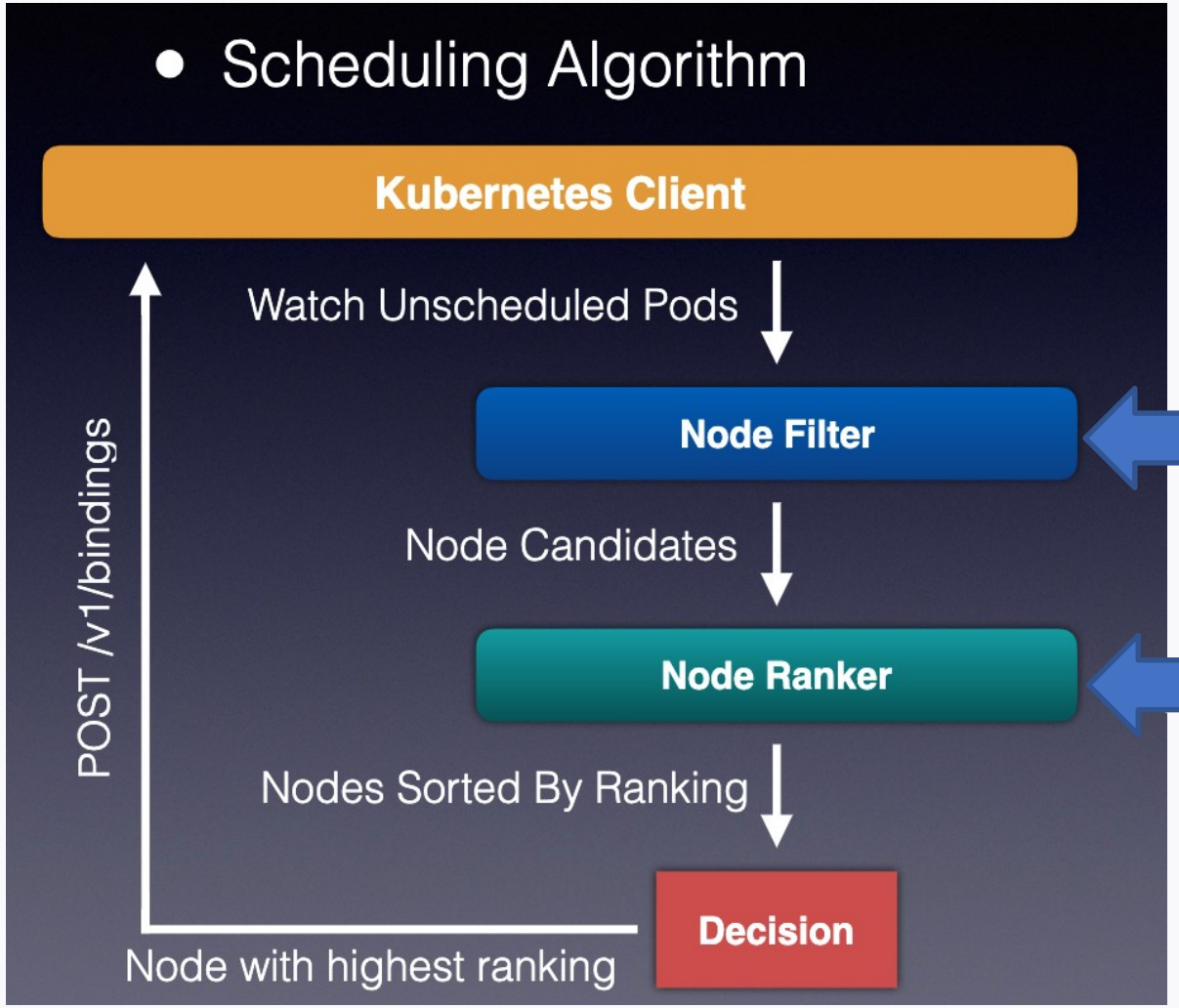


Kube-controller-manager

kube-controller-manager это контур управления, который наблюдает за общим состоянием кластера через apiserver и вносит изменения, пытаясь переместить текущее состояние в желаемое.



Kube-scheduler



Отфильтровать Impossible Node

- Готовность узла
- На узле должно быть достаточно памяти
- На узле должно быть достаточно CPU
- HostPort пода должен быть доступен
- …

Дать каждой ноде оценку и выбрать лучшую

- узлы «больше простаивают»
- реплики должны быть распределены
- Affinity / AntiAffinityPriority: «Я предпочитаю держаться подальше / размещаться вместе с некоторыми другими категориями модулей»
- узел предпочтительнее, если он имеет image
- утилизация баланс ноды

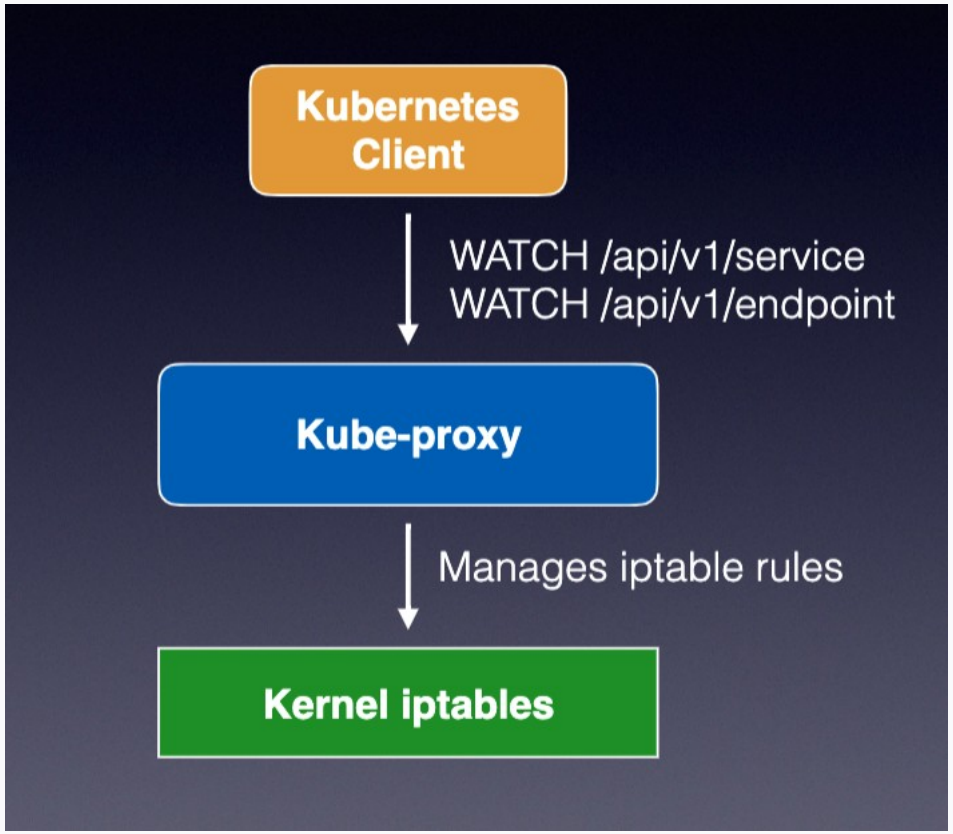
Service & Endpoint

Типы ресурсов:

- **Service** - абстрактный способ представить приложение, работающее на наборе Pods, в качестве сетевой службы.
- **Endpoint** – точка входа

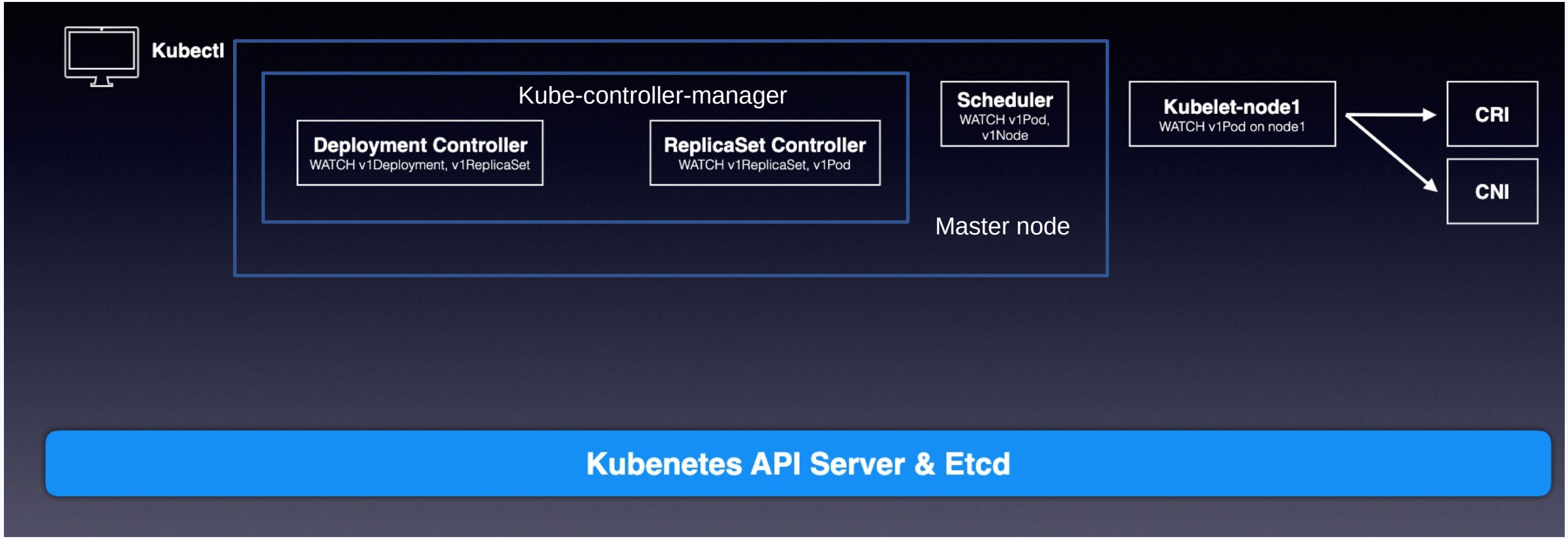
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Kube-proxy



- Дэмон на каждой ноде
- Следит за изменениями Service и Endpoint
- Управляет правилами iptable на хосте для достижения балансировки нагрузки службы (одна из реализаций)

Kubernetess Reaction Chain



Kubernetes REST API

- Каждая часть информации в Kubernetes представляет собой объект API (также известный как метаданные кластера)
  - Node, Pod, Job, Deployment, Endpoint, Service, Event, PersistentVolume, ResourceQuota, Secrets, ConfigMap, ...
  - Каждый объект API имеет UID и номер версии
- Микросервисы могут выполнять CRUD для объектов API через REST
- Желаемое состояние в кластере достигается за счет взаимодействия отдельных автономных объектов, реагирующих на изменения одного или нескольких API объект (ы), которые их интересуют

Область API

- Различные группы API. Обычно группа представляет собой набор функций, например batch, apps, rbac и т.д.
- Каждая группа имеет собственный контроль версий

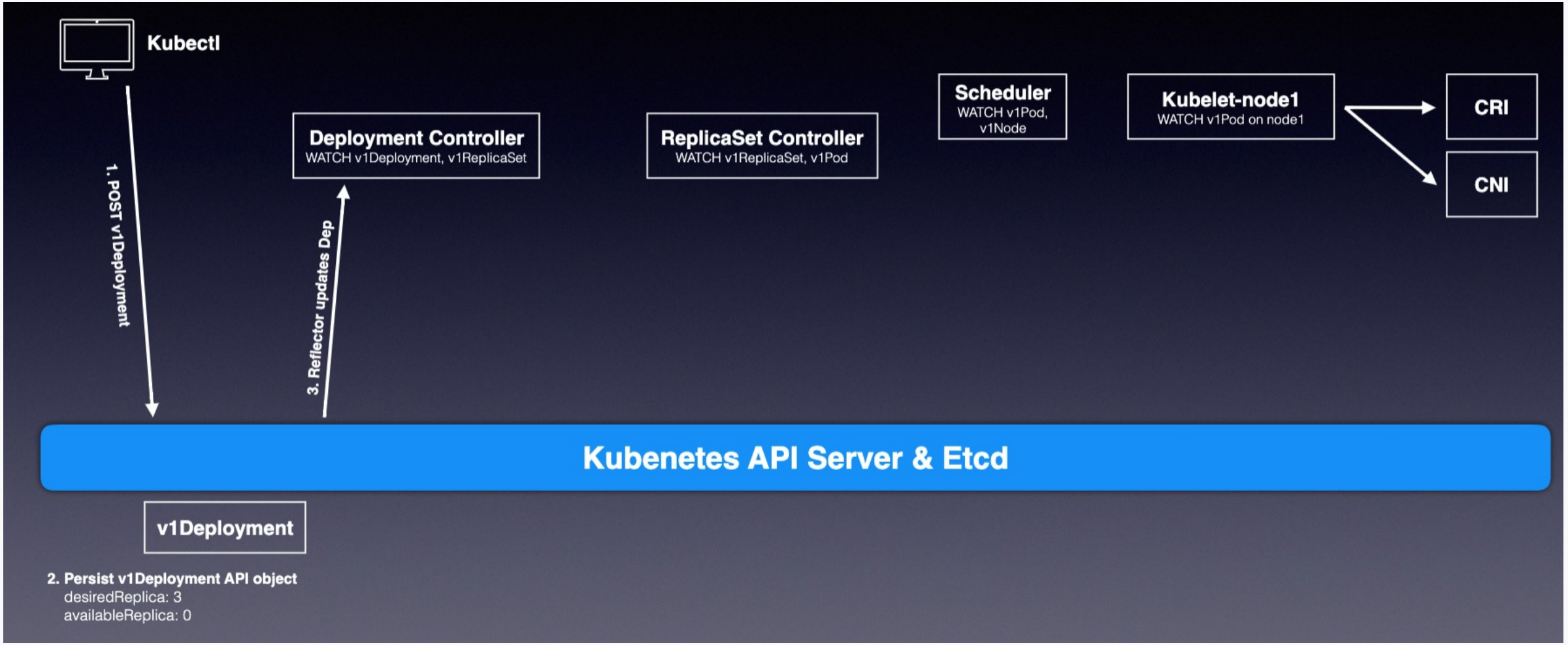
Kubernetes REST API

Определение объекта API

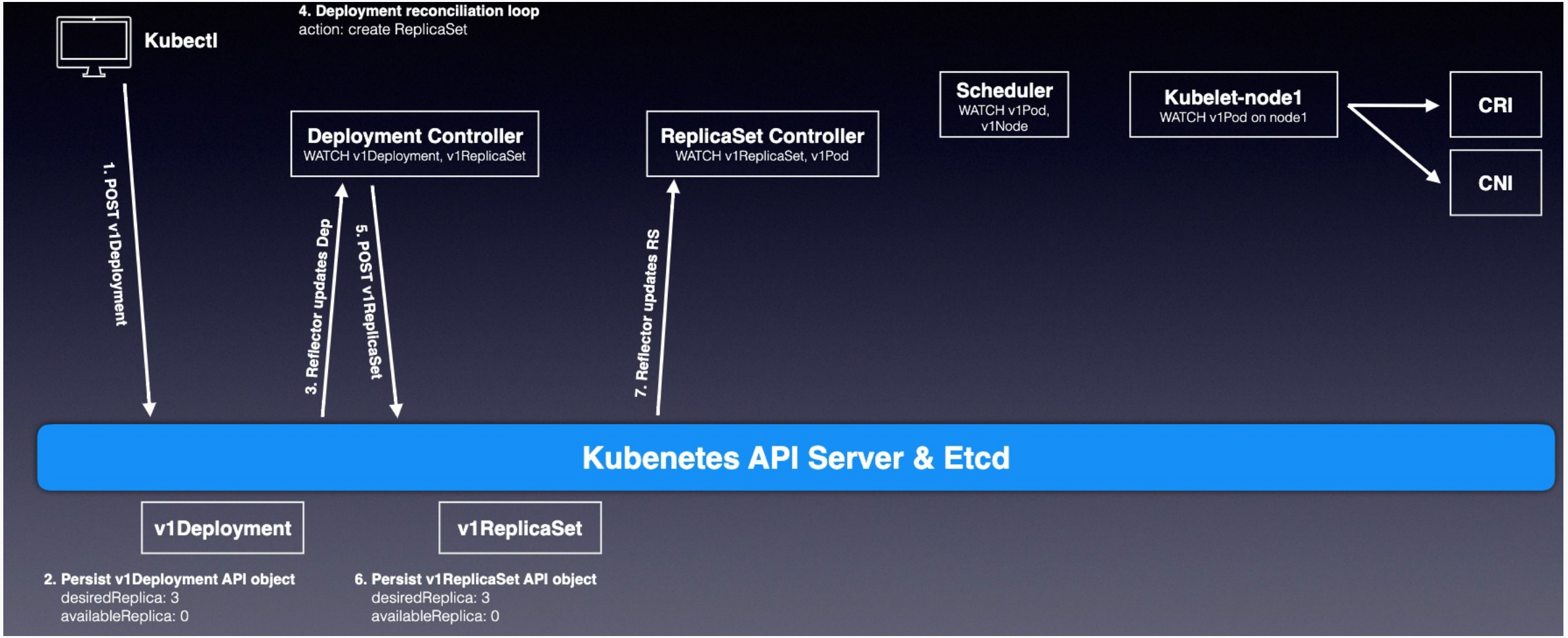
- 5 основных полей: **apiVersion, kind, metadata, spec** и **status** (несколько исключений, таких как v1Event, v1Binding и т. д)
- **Metadata** включает в себя name, namespace, label, annotation, version и т.д.
- **Spec** описывает желаемое состояние, в то время как **status** описывает текущее состояние

```
$ kubectl get jobs myjob --namespace default --output yaml
apiVersion: batch/v1
kind: job
metadata:
    name: myjob
    namespace: default
spec:
    (Describing desired “job” object, i.e. Pod template, how many runs, etc.)
status:
    (“job” object current status, i.e. ran # times already, etc.)
```

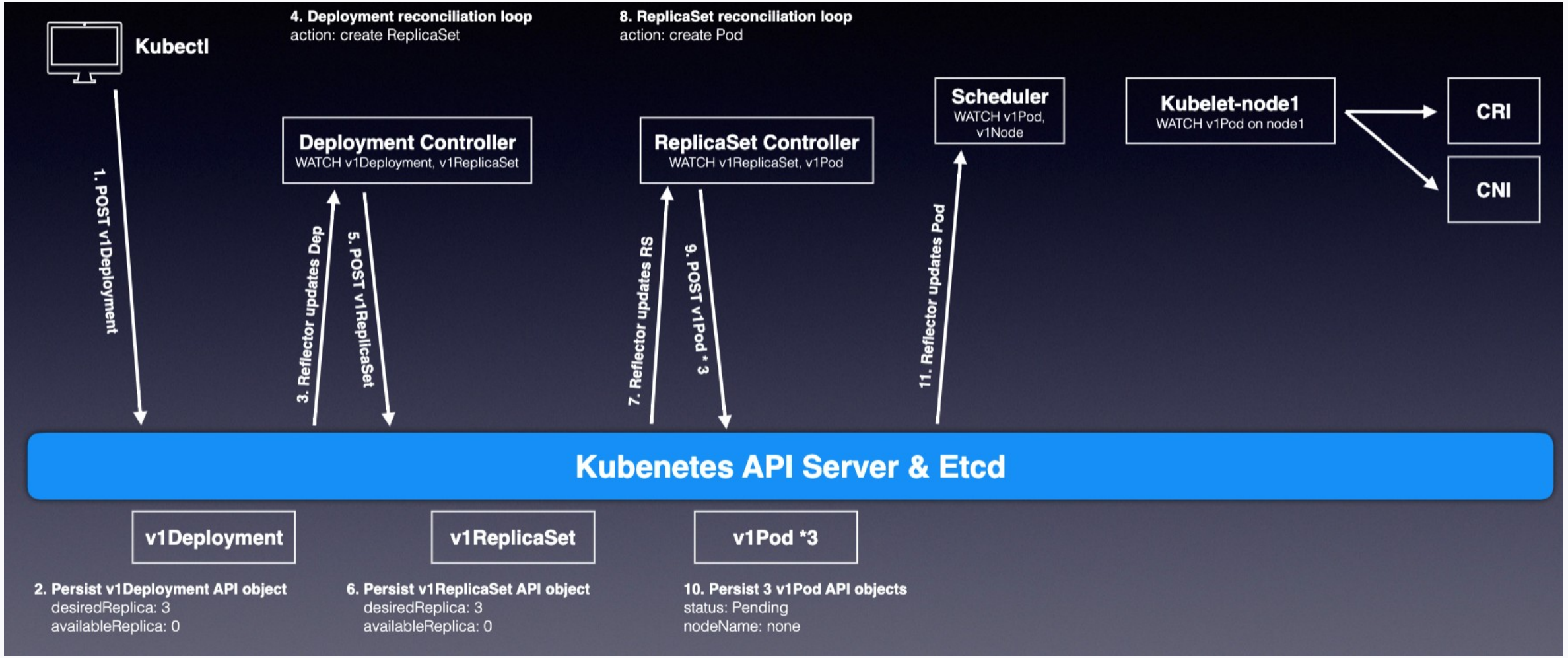
Kubernetess Reaction Chain



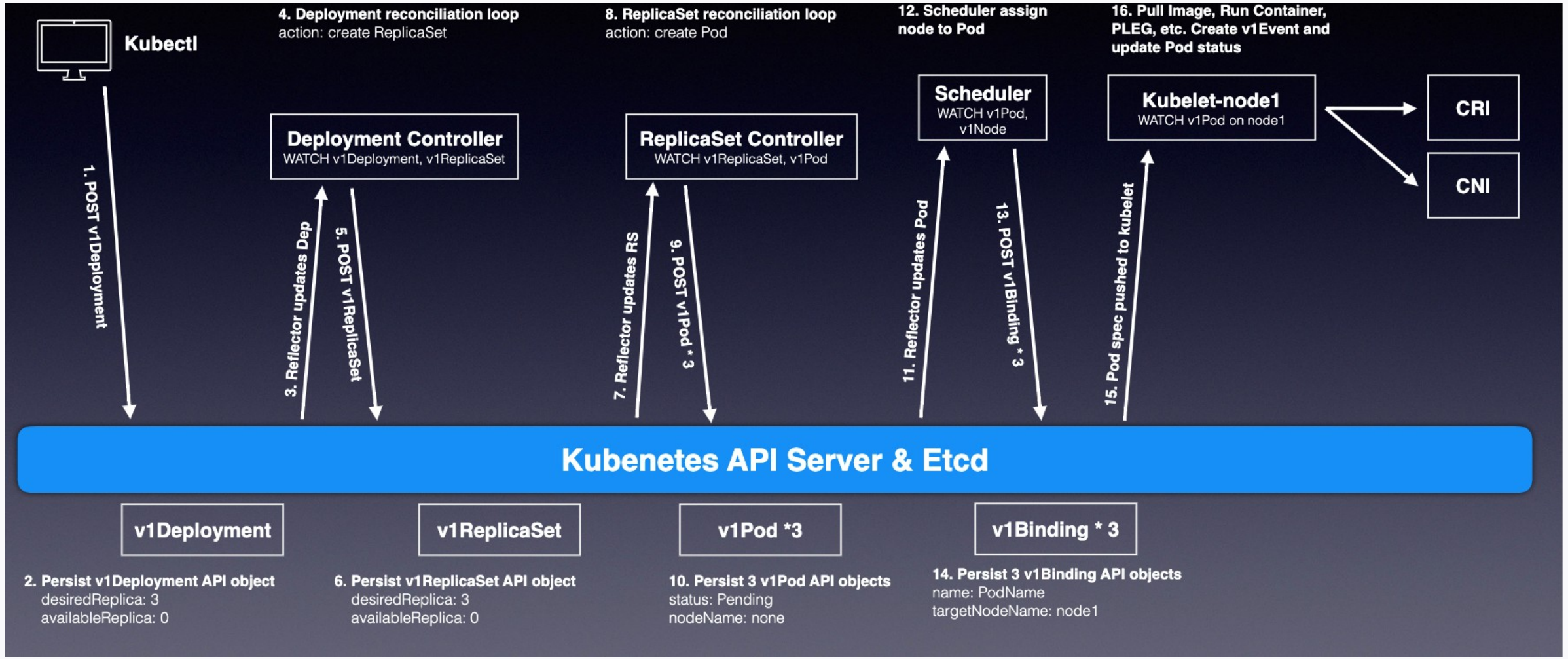
Kubernetess Reaction Chain



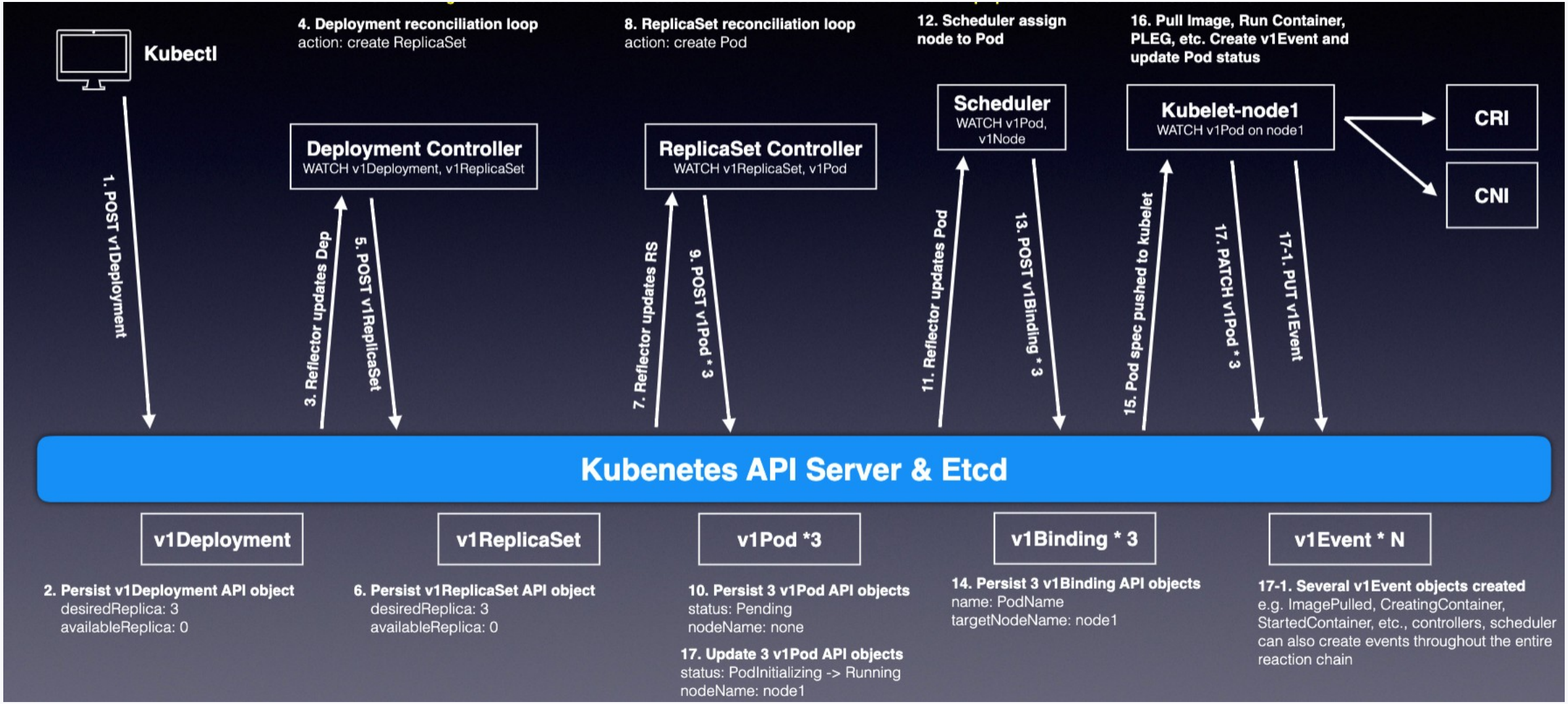
Kubernetess Reaction Chain



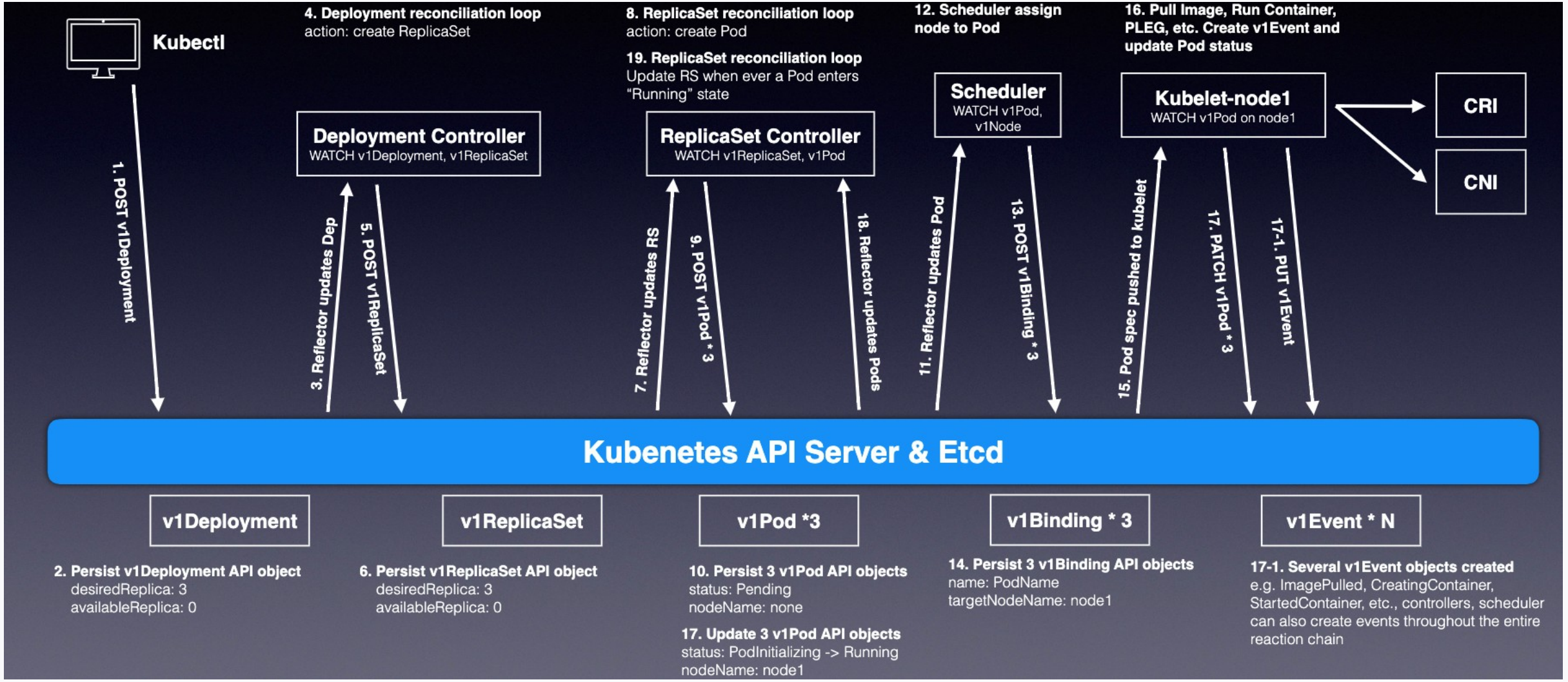
Kubernetess Reaction Chain



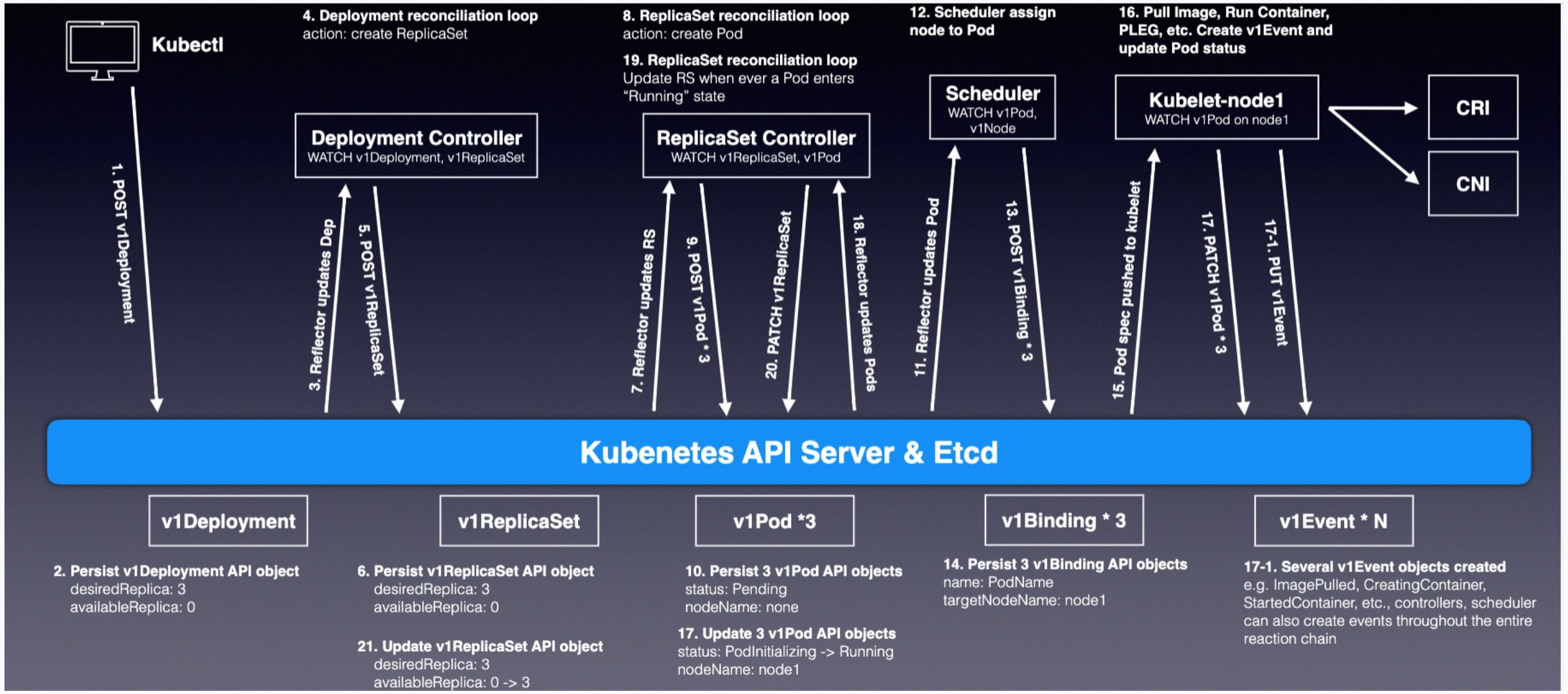
Kubernetes Reaction Chain



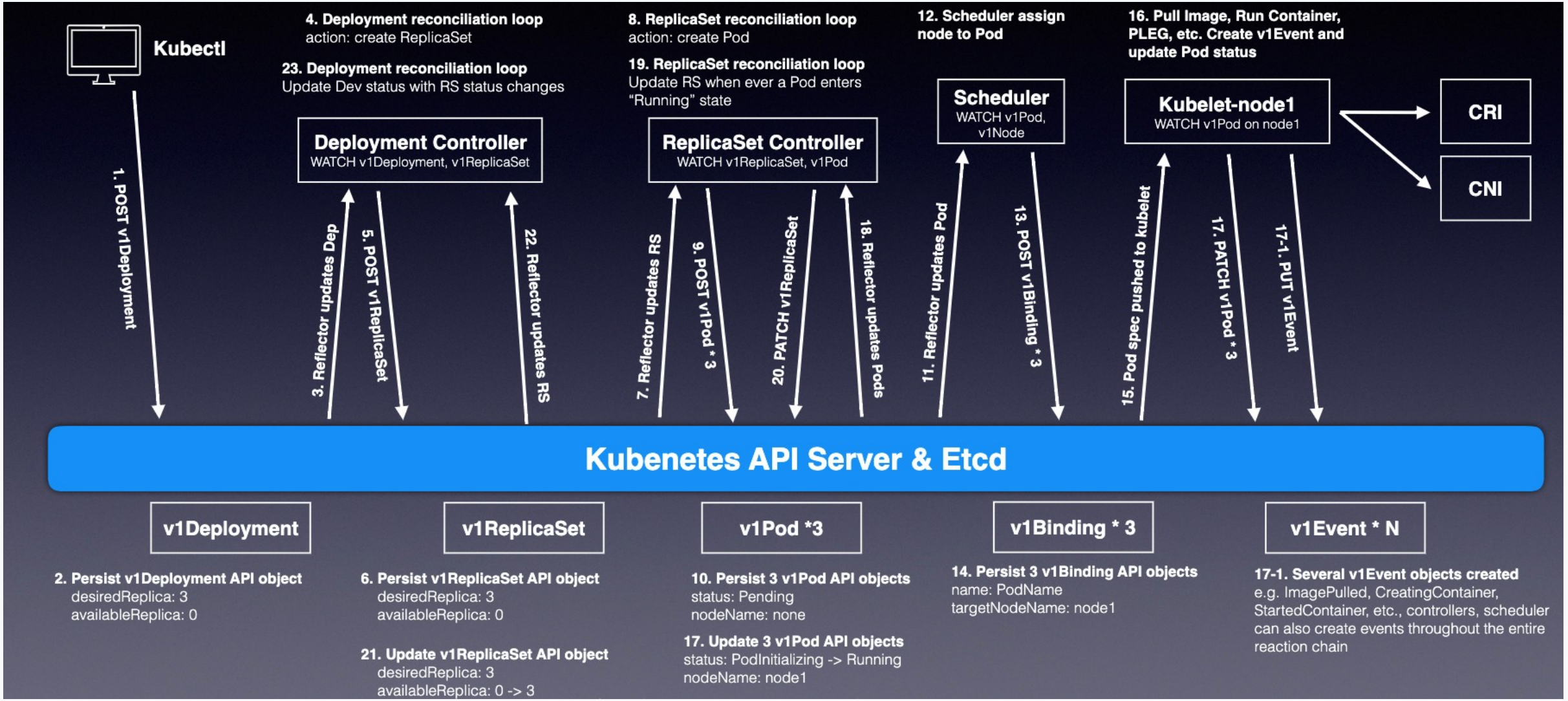
Kubernetes Reaction Chain



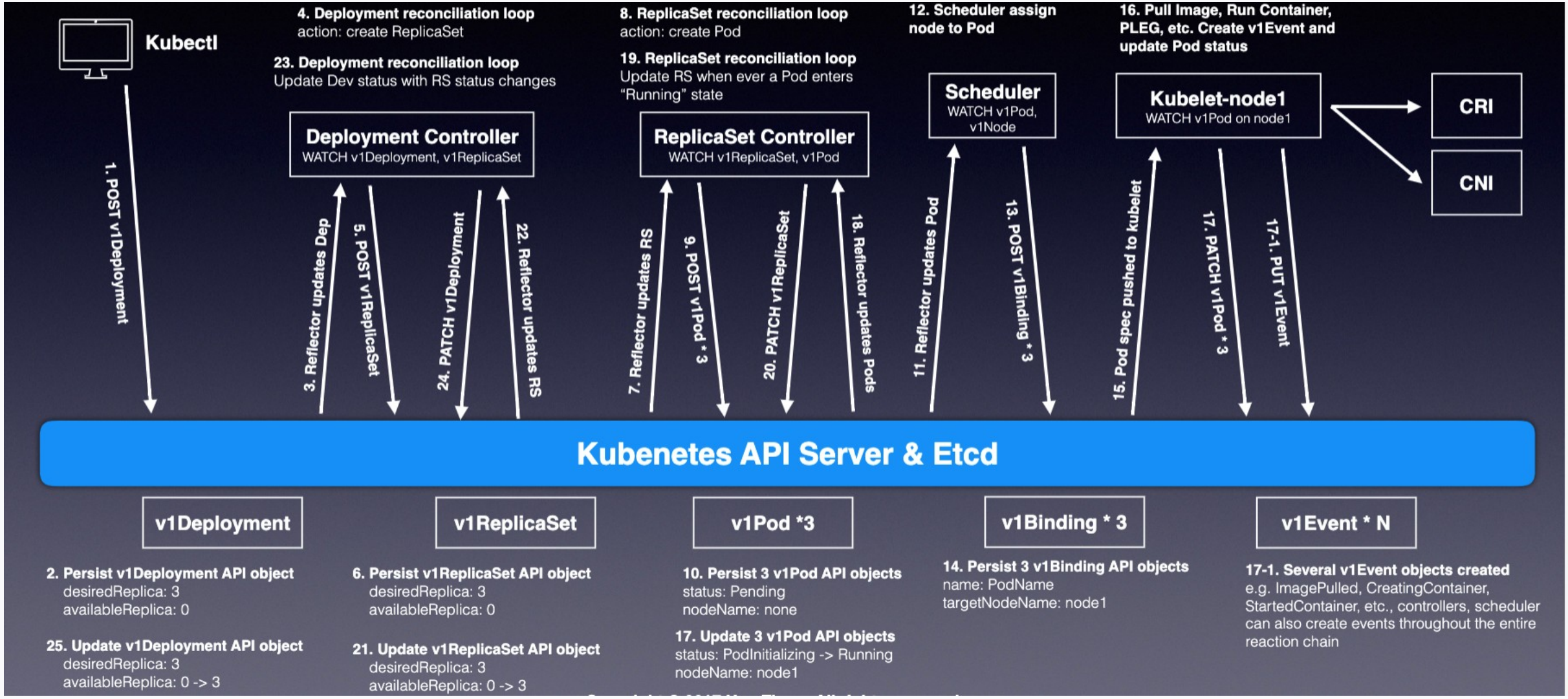
Kubernetes Reaction Chain



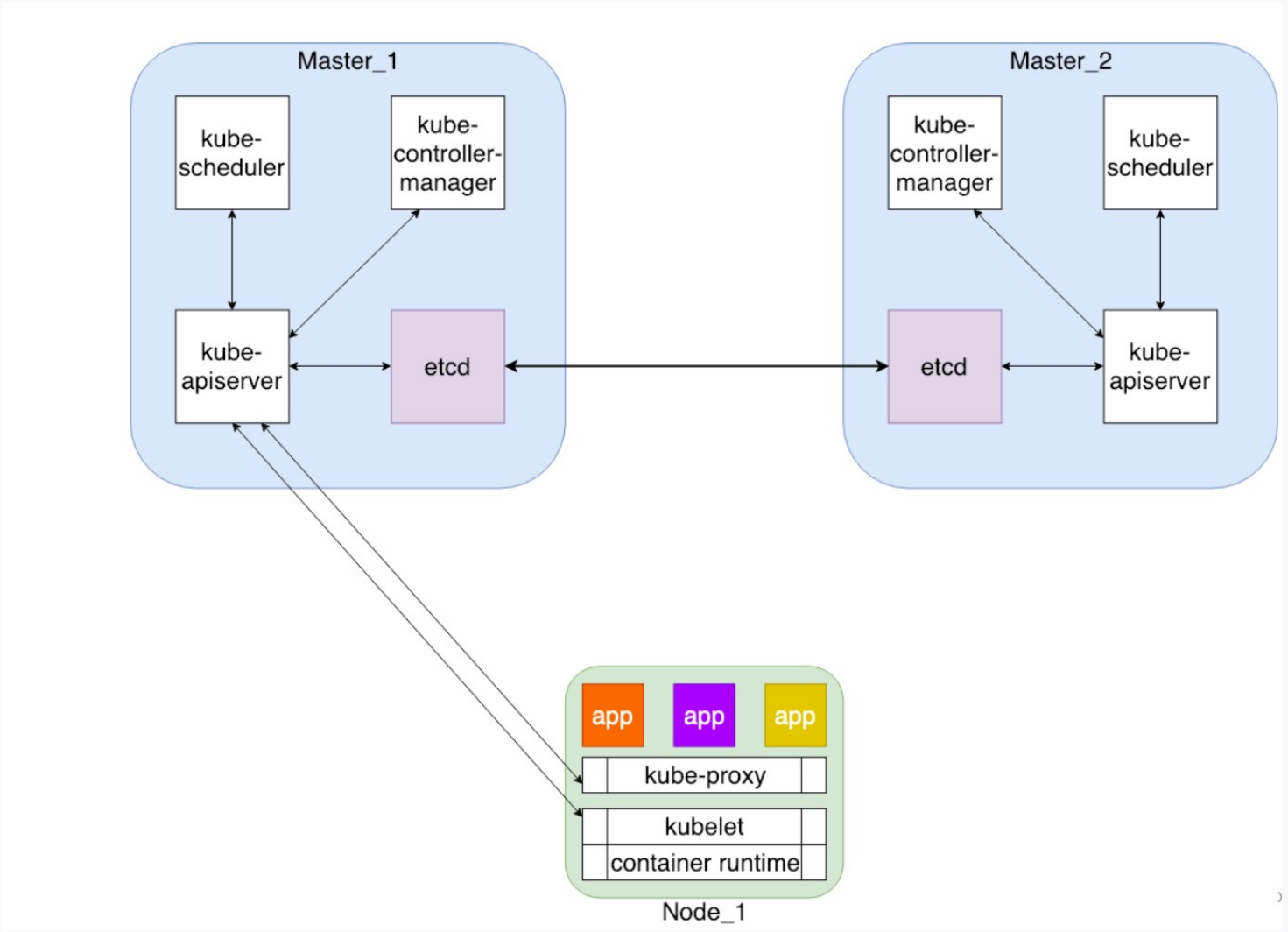
Kubernetes Reaction Chain



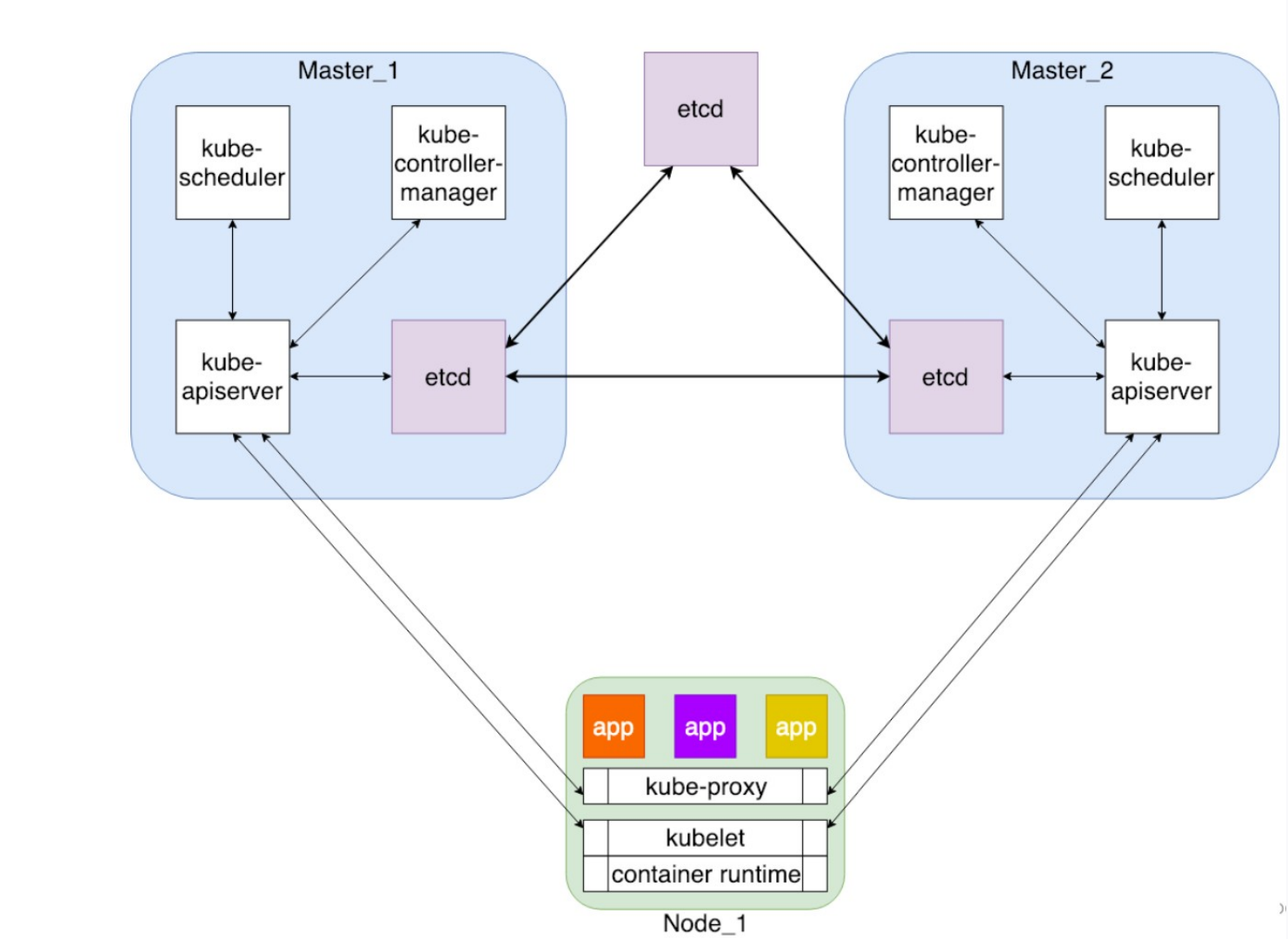
Kubernetes Reaction Chain



Архитектура Kubernetes



Архитектура Kubernetes



Архитектура Kubernetes

Addons

- Расширяют функционал Kubernetes
- Запускаются также, как и остальные сервисы и поды
- Взаимодействуют с Kubernetes через API

Примеры

- kube-dns/CoreDNS
- Сетевые плагины (Weave, Calico, Flannel)
- Dashboard/Weave Scope
- ...

kubectl | cluster

Кластер (cluster) - это:

- **server** - адрес kubernetes API-сервера
- **certificate-authority** - корневой сертификат (которым подписан SSL-сертификат API-сервера)
- **name** - имя для идентификации

```
clusters: #  ⬅️ Список кластеров
- cluster:
    certificate-authority: ca.crt
    server: https://35.198.140.134
  name: kube-cluster
```

kubectl | user

Пользователь (**user**) - это:

- Данные для аутентификации:
  - username + password (Basic Auth)
  - client key + client certificate
  - token
  - auth-provider config (например GCP)
- name (имя) для идентификации в конфиге

```
users: # ⬅️ Список пользователей и способов их авторизации
- name: kube-user
  user:
    client-certificate: kube-user.crt
    client-key: kube-user.key
```

kubectl | context

Контекст (**context**) - это:

- **cluster** - имя кластера из списка clusters
- **user** - имя пользователя из списка users
- **namespace** - область видимости по-умолчанию (не обязательно)
- **name** - имя контекста для идентификации

```
contexts:  # ⬅️ Список контекстов
- context:
    cluster: kube-cluster
    user: kube-user
  name: kube-context
```

kubectl

Пример конфигурации kubectl

```
apiVersion: v1
clusters:                    # ⬅️ Список кластеров
- cluster:
    certificate-authority: ca.crt
    server: https://35.198.140.134
  name: kube-cluster
contexts:                    # ⬅️ Список контекстов
- context:
    cluster: kube-cluster
    user: kube-user
  name: kube-context
current-context: kube-context # ⬅️ Текущий контекст
kind: Config
preferences: {}
users:                       # ⬅️ Список пользователей и способов их авторизации
- name: kube-user
  user:
    client-certificate: user.crt
    client-key: user.key
```

Порядок конфигурации kubectl

Обычно порядок конфигурирования `kubectl` следующий:

```
# Указать кластер:
kubectl config set-cluster … cluster_name

# Указать данные пользователя (credentials):
kubectl config set-credentials … user_name

# Создать контекст:
kubectl config set-context context_name --cluster=cluster_name --user=user_name

# Использовать контекст:
kubectl config use-context context_name
```

Команды kubectl

kubectl Cheat Sheet

```
# Создать ресурс из манифеста
kubectl create -f manifest.yml
kubectl apply -f manifest.yml
kubectl apply -f link
kubectl apply -f directory/

# Получить список ресурсов
kubectl get pods

# Получить описание ресурса
kubectl describe pod POD_NAME
```

Рефлексия