

# Kubernetes

Попов Матвей

# План

- Что это такое?
- Как оно устроено?
- Какие есть примитивы?
- Как они работают?
- Пробуем запустить локально
- Дополнительные инструменты



**kubernetes**

**Что же это такое?**

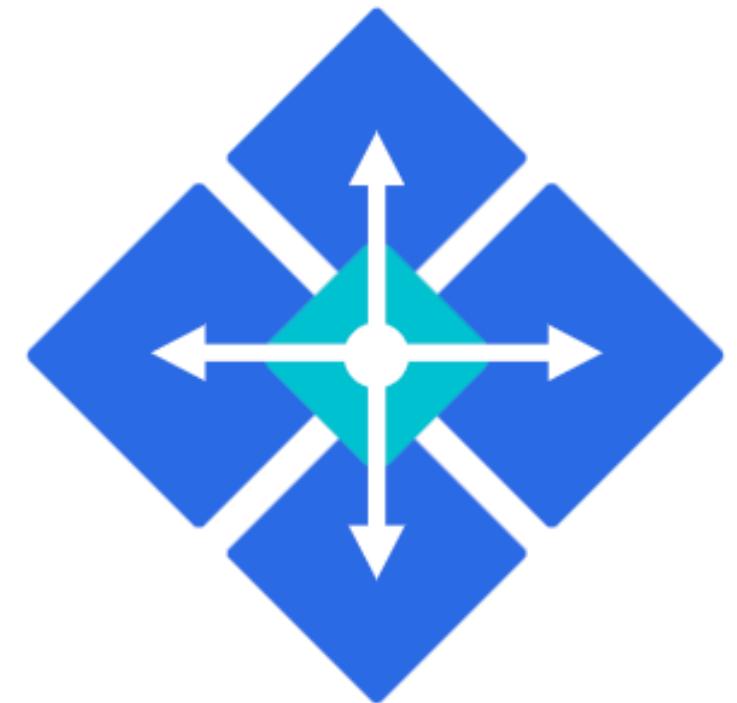
# Kubernetes

Это портативная расширяемая **платформа с открытым** исходным кодом **для управления контейнеризованными** рабочими нагрузками и **сервисами**, которая облегчает как **декларативную настройку**, так и **автоматизацию**

# Краткие факты:

- Управляет контейнерами
- Разрабатывалась Google с 2008 года
- Опубликована в 2014
- С 2015 имеет стабильную версию
- Написан на Go

**Еще факты**

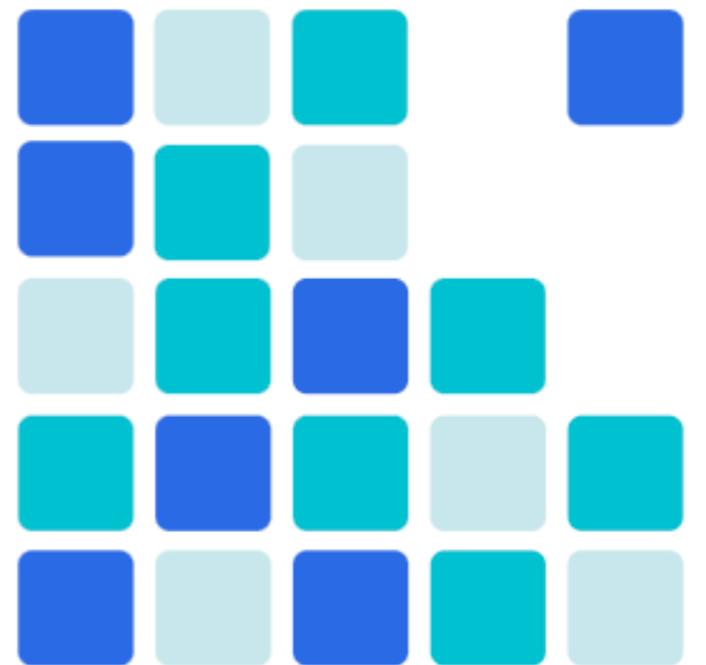


## Масштабируемость любого уровня

Разработанный на тех же принципах, которые позволяют Google запускать миллиарды контейнеров в неделю, Kubernetes может масштабироваться без увеличения вашей технической команды.

## Бесконечная гибкость

Будь то локальное тестирование или работа в корпорации, гибкость Kubernetes растёт вместе с вами, обеспечивая бесперебойную и простую доставку приложений независимо от сложности ваших потребностей.



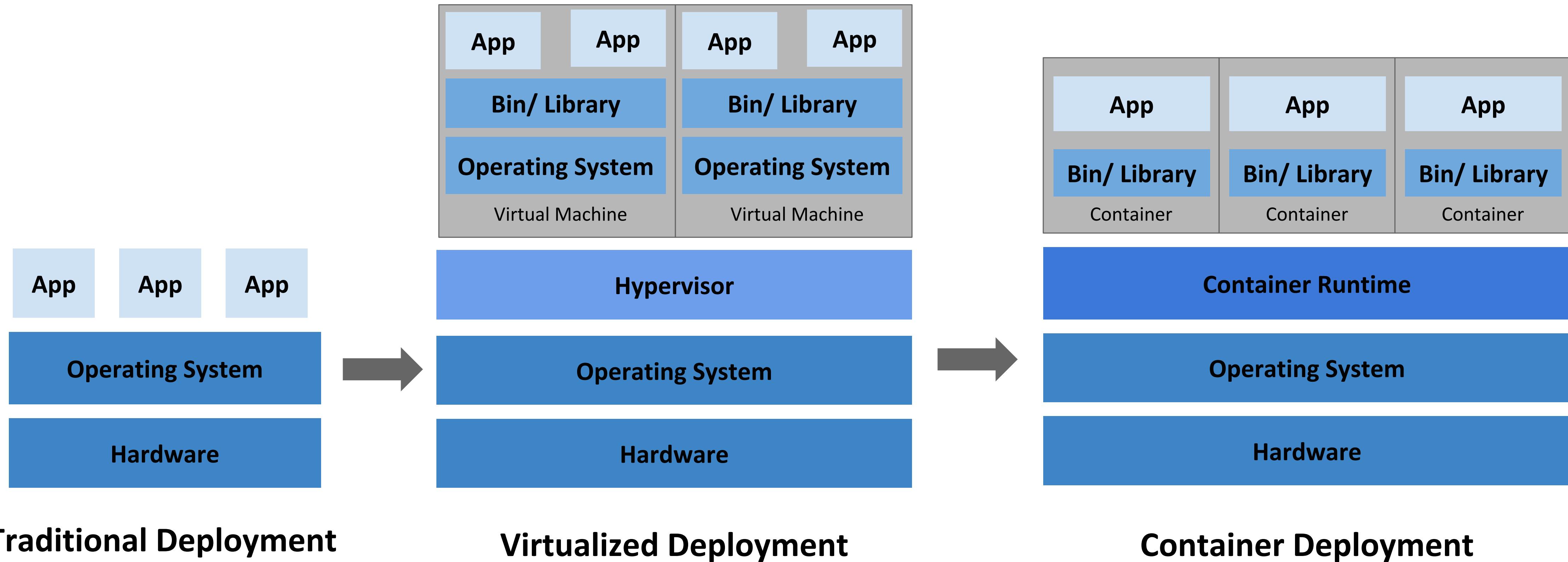


## Может быть запущен где угодно

Kubernetes — это проект с открытым исходным кодом, который даёт вам полную свободу воспользоваться преимуществами локальной, гибридной или публичной облачной инфраструктуры, позволяя без усилий перераспределять рабочую нагрузку по мере необходимости.

**Посмотрим в прошлое**

**Как менялась организация запуска  
приложения с течением временем?**



# Traditional Deployment:

- Запуск на физических серверах
- Нет способа ограничить границы ресурсов для приложения
- Запуск каждого нового приложения на отдельном сервере
- Поддержка большего количества физических серверов

# Virtualised Deployment:

- Запуск нескольких виртуальных машин на одном физическом сервере
- Изоляция приложений между ВМ
- Эффективное использование ресурсов
- ВМ - полноценная машина

# Container Deployment:

- Похожи на виртуальные машины
- Совместное использование ресурсов
- Очень гибкие, легкие и переносимые

# Еще факты про контейнеры:

- Их нужно мониторить
- Между ними нужно распределять нагрузку
- Нужно автоматически развертывать и откапывать
- Нужно управлять конфигурацией

**Вот тут Kubernetes приходит  
на помощь!**



# А что такое контейнеры?



# Контейнеризация

Метод **виртуализации**, при котором ядро операционной системы поддерживает **несколько** экземпляров **пространства пользователя** вместо одного.

**Какие есть реализации?**



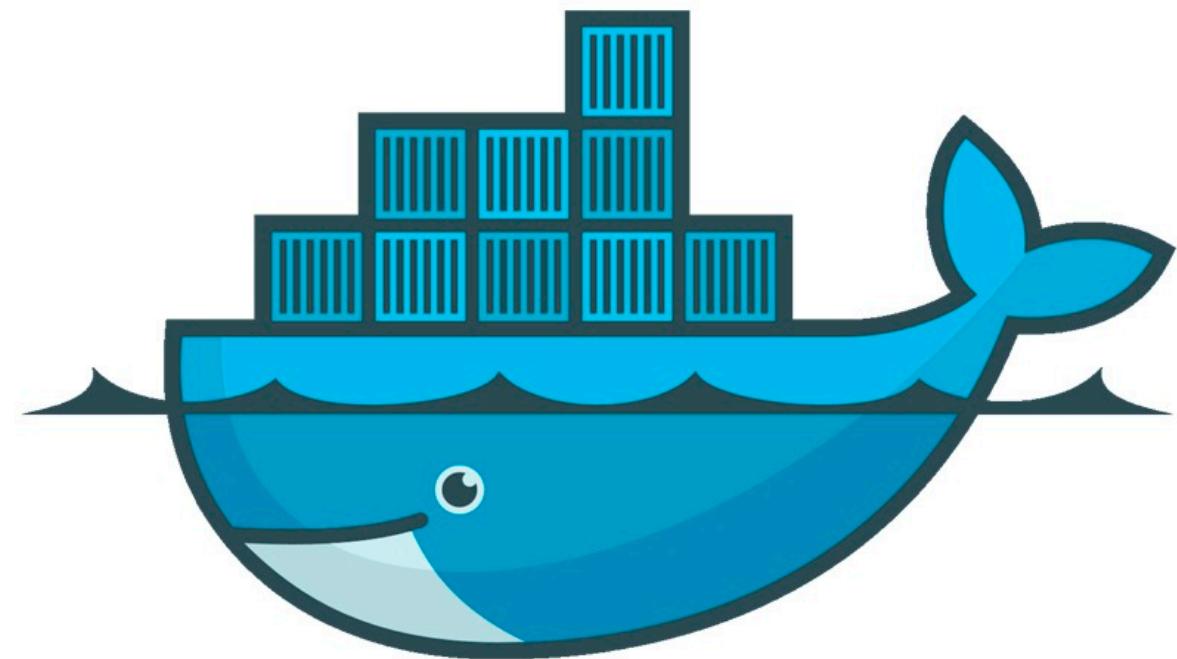
Virtuozzo



solaris

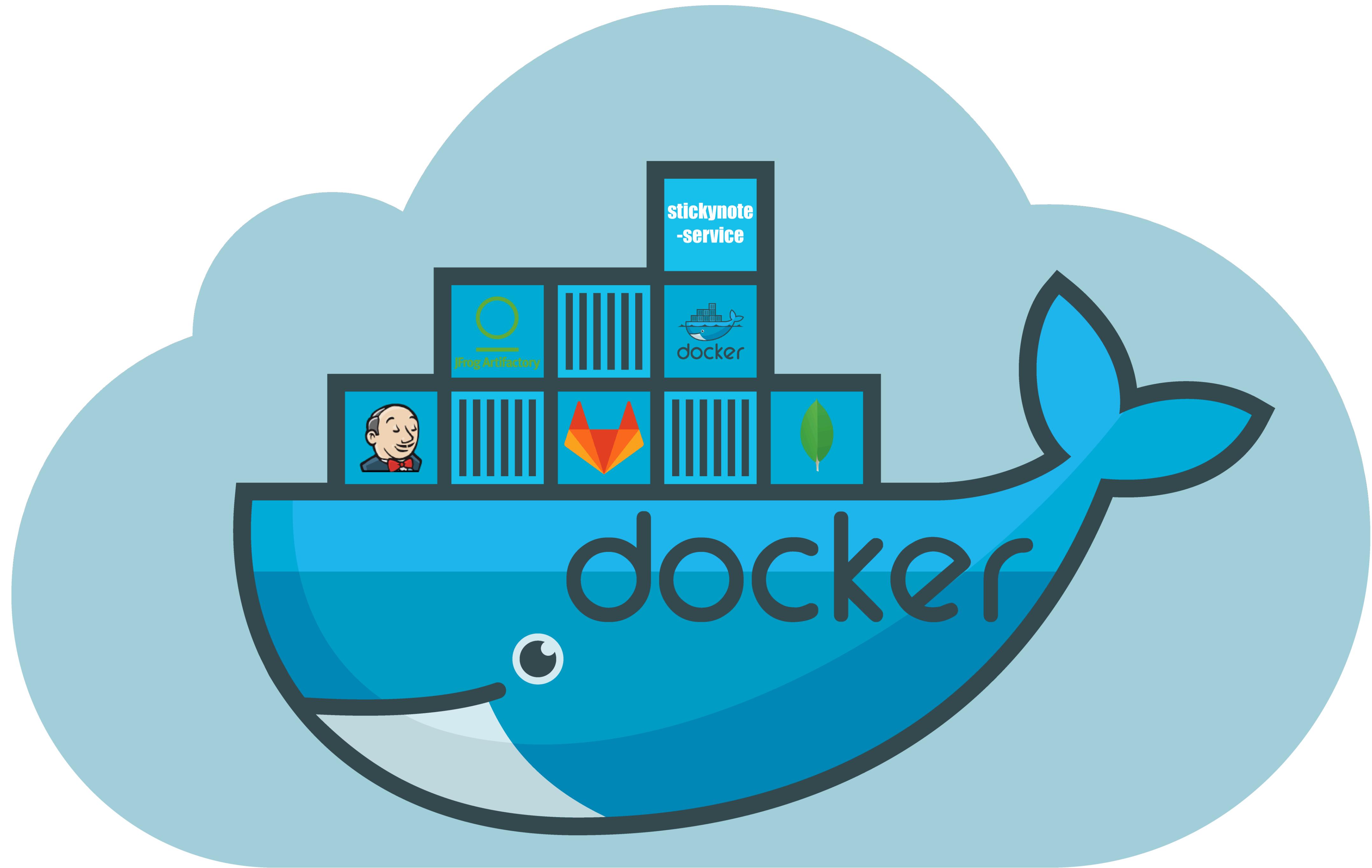


OpenVZ  
Linux Containers



docker

**Рассмотрим самое  
популярное решение**



# Краткий(очень) экскурс в Docker

**Что это?**

# Очень простыми словами:

- ПО для виртуализации
- Упрощает разработку приложений и их развертывание
- Упаковывает приложения со всеми необходимыми зависимостями, конфигурациями, системными инструментами и runtime

# Контейнер

Стандартизированная **абстракция**, которая имеет все что необходимо для запуска приложения

**Посмотрим на базовые  
примитивы**

# Docker Image

- Описание как и с чем должен быть запущен контейнер
- Содержит описание ОС
- Все необходимые зависимости
- Описание как запустить приложение

**Как можно описать  
инструкцию по запуску?**

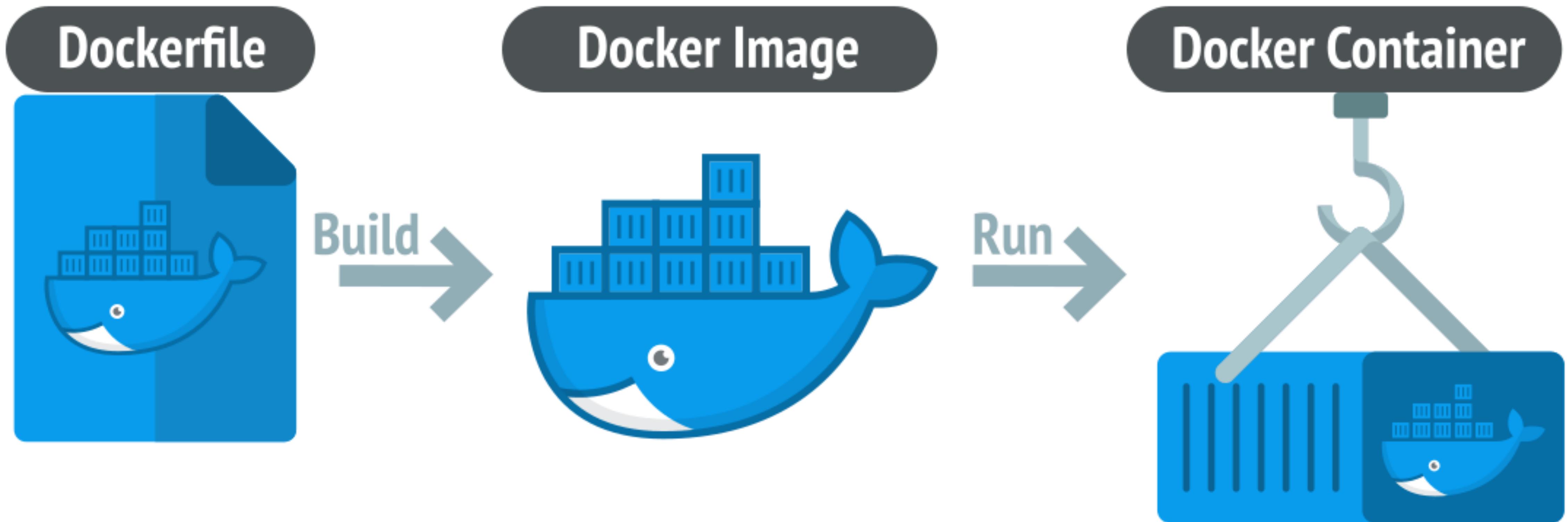
**С помощью Dockerfile**

```
1      # базовый образ
2 ▷  FROM ubuntu:20.04
3
4      # установка желаемых пакетов и зависимостей
5      RUN apt-get update
6      RUN apt-get install -y wget && echo "We installed wget!"
7
8      # создание каталога /mytest
9      WORKDIR /mytest
10
11     # создание файла testfile
12     RUN touch testfile
13
14     # ls и echo будут выполняться при запуске контейнера
15     CMD ls && echo "Show me mytest"
16
```

# Docker Container

- Работающий экземпляр Docker Image
- Из одного Docker Image может быть запущено много Docker Container

# **Итог**



# Docker Registry

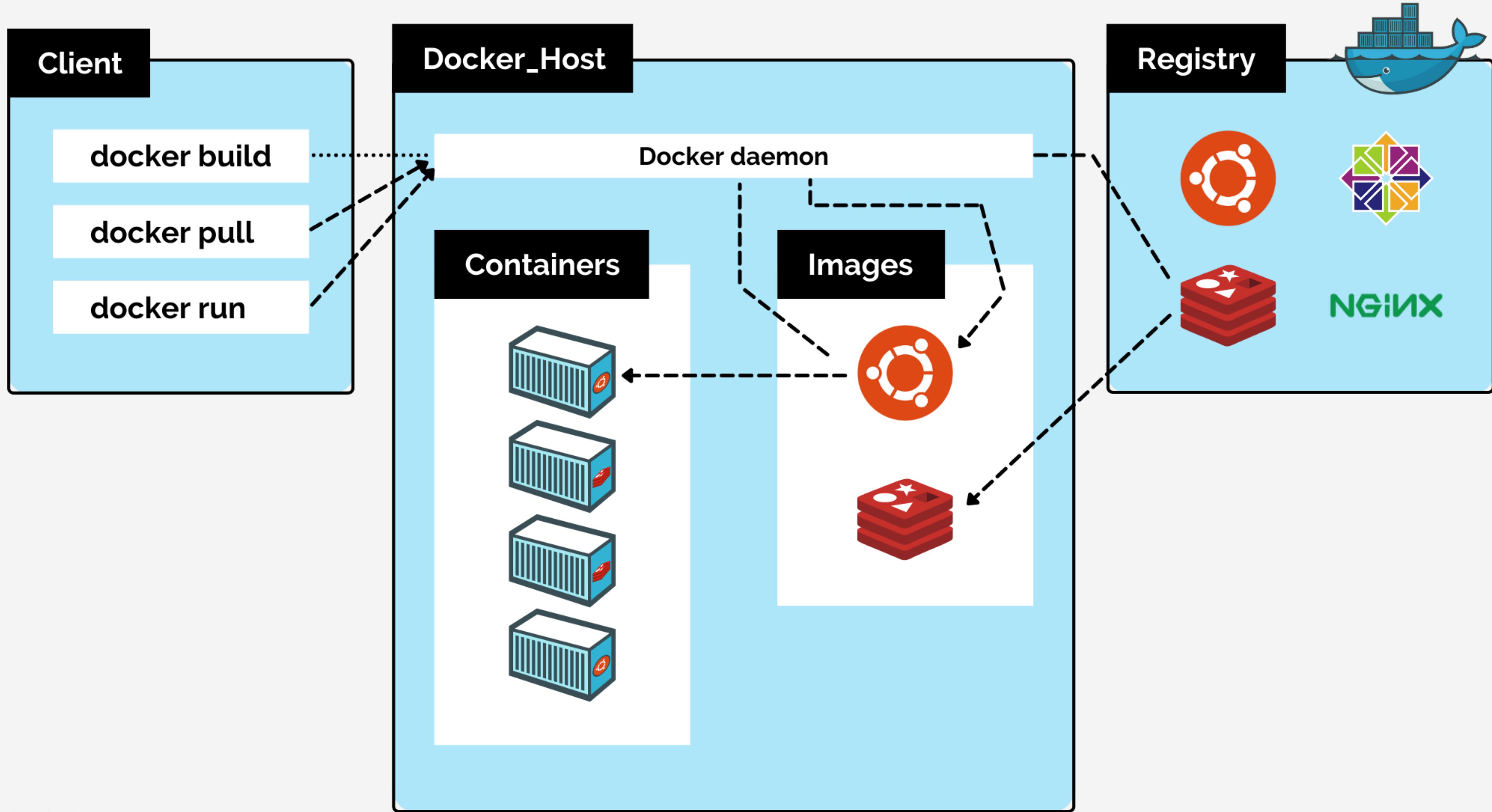
- Мы знаем как получить Docker Container из Docker Image
- Но как получить Docker Image?

**Они хранятся в Docker  
Registries!**

**Docker Registry - хранилище с  
версионированными докер-образами**

**Как это может быть полезно?**

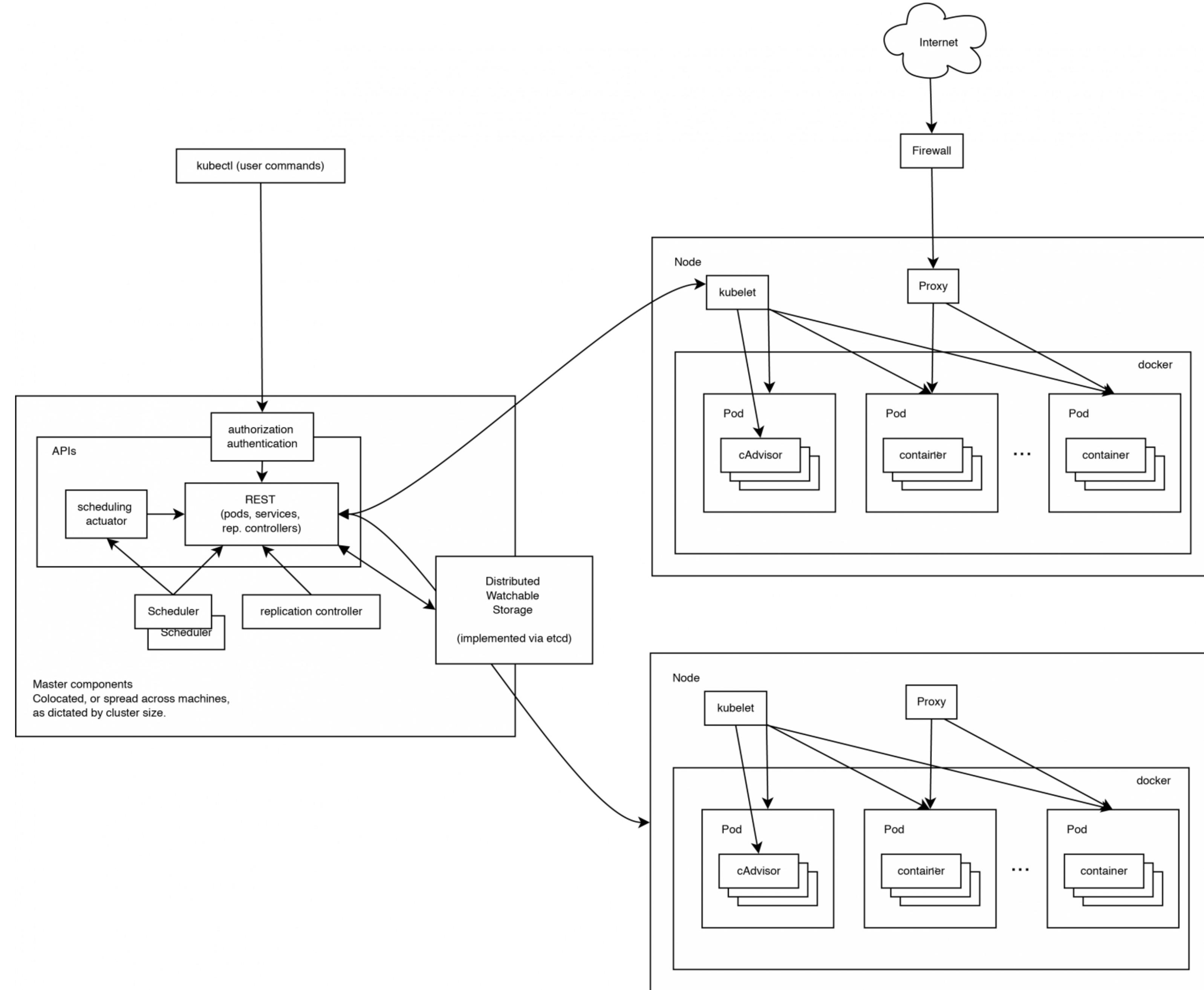
- Собираем на рабочем ПК образ и отправляем в Registry
- На сервере забираем образ из Registry и запускаем



**С полученными знаниями  
вернемся к Kubernetes**

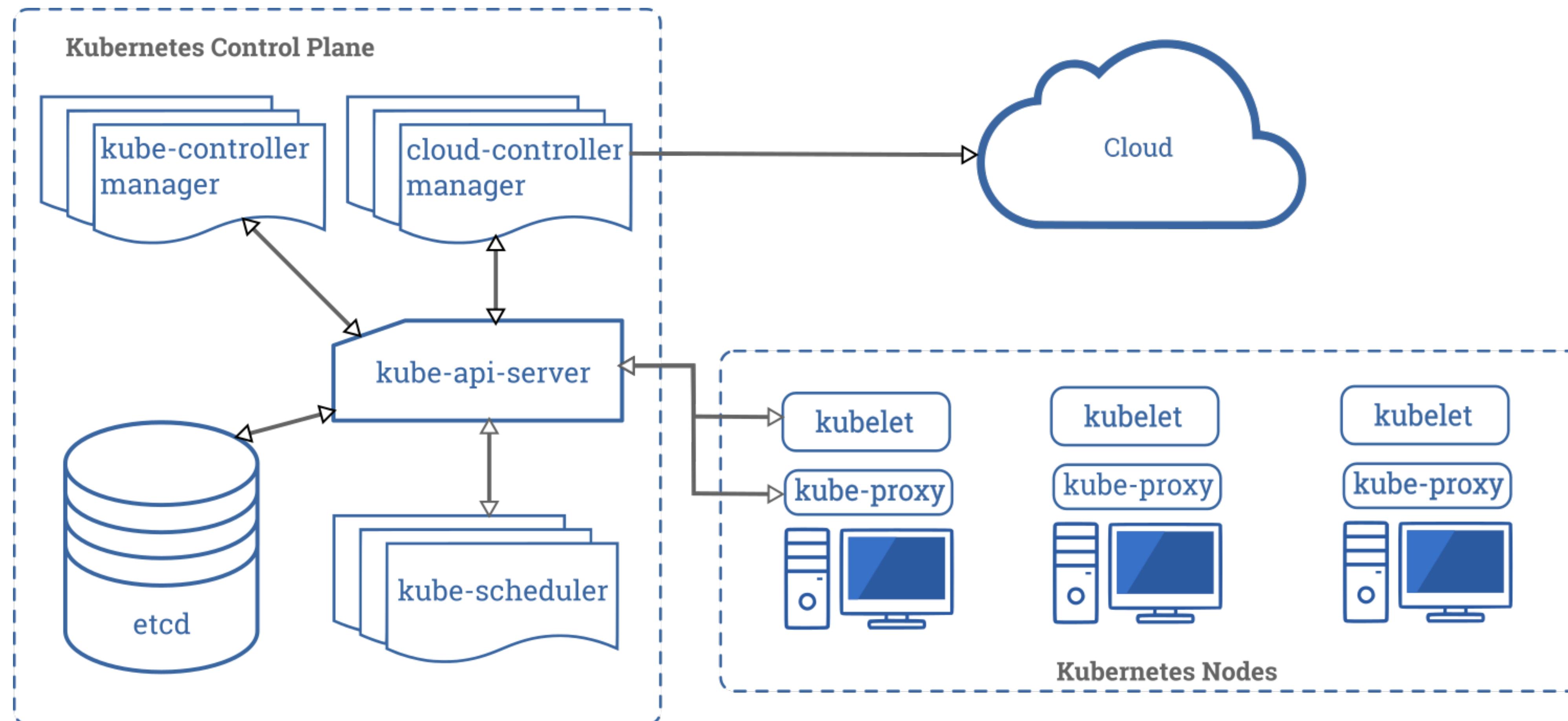
**Какие есть компоненты у k8s?**

- Основной компонент это Cluster
- Каждый Cluster состоит из Node
- Node существуют двух типов:
  - Worker Node - сервер на котором запускаются и работают контейнеры
  - Master Node - сервер который управляет всеми Worker Node
- Все команды взаимодействия от клиентов всегда идут к Master Node

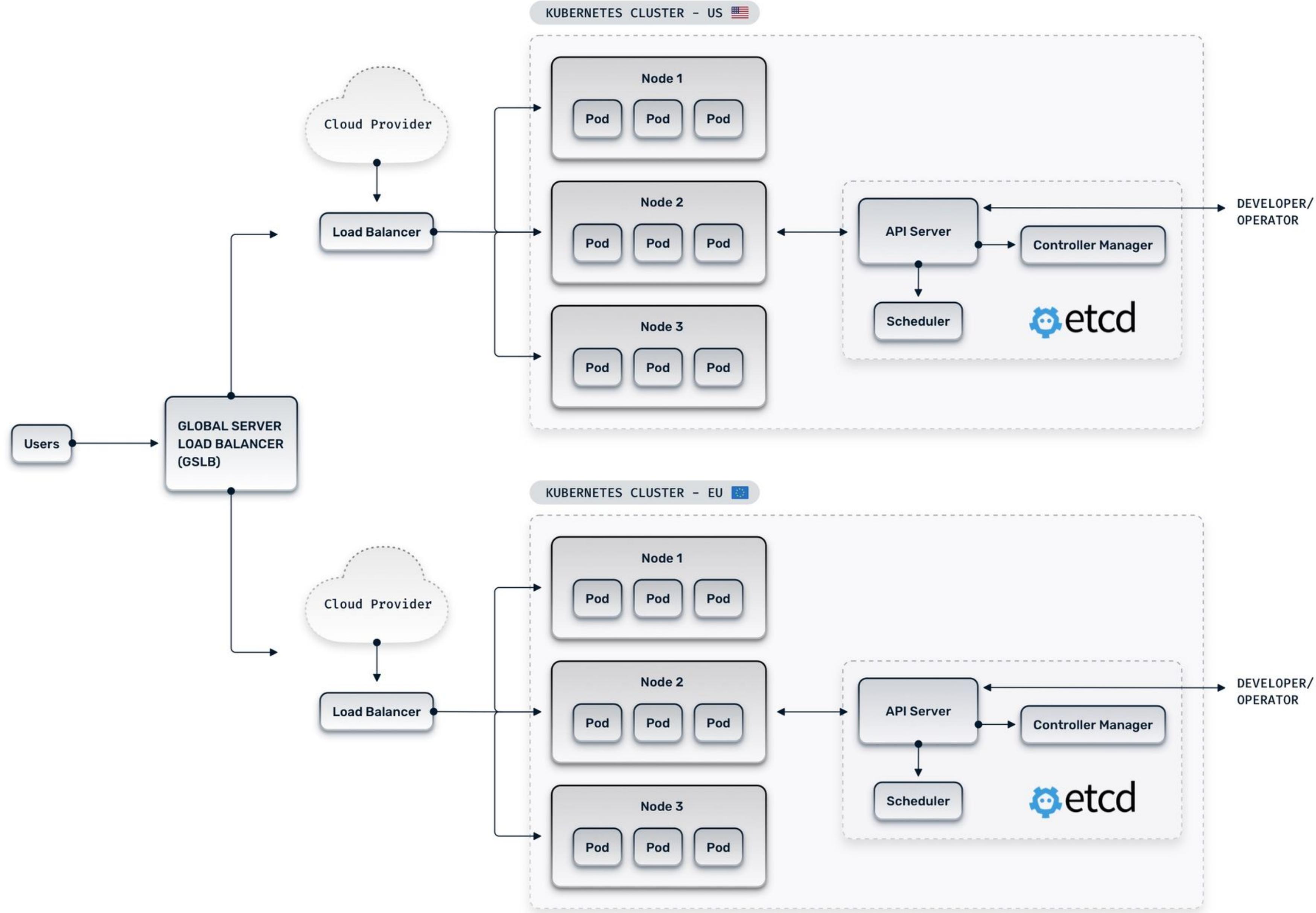


**Как-то сложновато....**

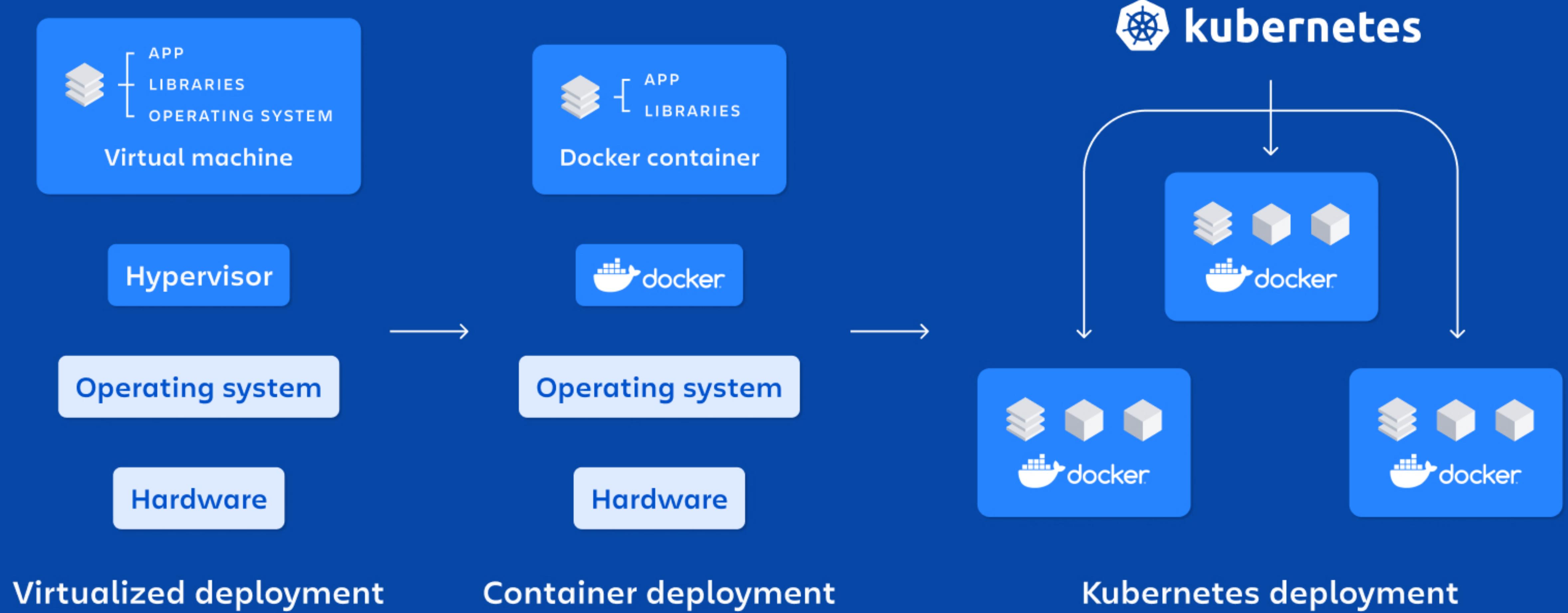
**Есть схема попроще**



**Как обычно выглядит Cluster?**



**Можно провести аналогию с  
различными Deployment**



**Какие есть пользовательские  
компоненты?**

# Самые основные:

- Container
- Pod
- Deployment
- Service
- Ingress
- Nodes
- Cluster

# А также:

- DaemonSet
- StatefulSet
- ReplicaSet
- DeploymentSet
- Secret
- Job
- CronJob
- PV
- SVC
- LoadBalancer
- ConfigMaps
- Vertical Pod AutoScaller
- Horizontal Pod AutoScaller
- ...

# **Начнем с Pod**

**Что это такое?**

**Pod представляет собой запрос на запуск одного или более контейнеров на одном узле(Node)**

Pod - это **один и единственный**  
объект в k8s, который приводит к  
**запуску контейнеров**

# **Как определить род?**

**С помощью manifest файла**

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: KtorExamplePod
5   labels:
6    environment: testing
7    platform: kotlin-ktor
8  spec:
9    containers:
10   - name: mipt-kotlin-ktor
11     image: kotlinktorexample:v1
12     ports:
13       - containerPort: 8080
14
```

# Deployment

**Что это такое?**

**Это абстракция Kubernetes, управляющая  
жизненным циклом приложения и  
описывающая особенности его запуска**

# Какие у него задачи?

- Сколько реплик нужно запустить?
- Сколько ресурсов нужно выделить?
- Какие health check-и использовать?
- Какие метки и селекторы использовать?
- Какие стратегии обновления?

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mipt-kotlin-ktor-deployment
5  labels:
6    owner: Ferum-bot
7    environment: testing
8    platform: kotlin-ktor
9    namespace: default
10 spec:
11   replicas: 2
12   progressDeadlineSeconds: 120
13   strategy:
14     type: RollingUpdate
15     rollingUpdate:
16       maxUnavailable: 1
17   selector:
18 -:
19     matchLabels:
20       environment: testing
21       platform: kotlin-ktor
22   template:
23 -:
24     metadata:
25       labels:
26         environment: testing
27         platform: kotlin-ktor
28   spec:
29     containers:
30       - name: mipt-kotlin-ktor
31         image: kotlinktorexample:v1
32         readinessProbe:
33           httpGet:
34             port: 8080
35             path: /metrics
36             initialDelaySeconds: 40
37             periodSeconds: 20
38         ports:
39           - containerPort: 8080
40             hostPort: 8080
41         env:
42           - name: SOME_ENV
43             value: "Hello from deployment"
```

# Service

**Что это такое?**

**Service – это абстракция,  
определяющая набор методов и доступ  
к ним**

**Есть несколько типов**

**Они нужны, чтобы указать, где будет размещен  
наш сервис, будет ли он доступен снаружи или  
только внутри, используются типы сервиса**

# ClusterIP

Присваивает IP в сервисной сети, который доступен только  
**внутри** кластера

# NodePort

Предоставляет **порт на внешнем ip** самого узла (Node), сервис будет **доступен из вне** по выделенному порту на **каждом узле**(Node). При этом создается и внутренний сервис ClusterIP.

# LoadBalancer

Используется для **внешних облачных балансировщиков**, таких как Google Cloud, которые имеют **своего провайдера**. Сервис будет доступен через внешний **балансировщик** вашего провайдера, при этом создаются NodePort с портами, куда будет приходить трафик от провайдера и ClusterIP.

**Как выглядит manifest файл?**

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: mipt-kotlin-ktor-load-balancer
5   labels:
6     owner: Ferum-bot
7     environment: testing
8     platform: kotlin-ktor
9 spec:
10   type: ClusterIP
11 → selector:
12   environment: testing
13   platform: kotlin-ktor
14   application: mipt-ktor-example-app
15 ports:
16   - port: 8080
17     targetPort: 8080
18     protocol: TCP
19     name: mipt-ktor-example-listner
20
```

✓ 5 ^

# Ingress

**Ingress - это ресурс для добавления правил маршрутизации трафика из внешних источников в службы в кластере kubernetes**

**Будет проще понять на  
примере**

```
1 apiVersion: networking.k8s.io/v1
2 kind: Ingress
3 metadata:
4   name: testing-ingress
5 annotations:
6   kubernetes.io/ingress.class: "nginx"
7   cert-manager.io/cluster-issuer: "letsencrypt"
8 spec:
9   tls:
10    - hosts:
11      - sub-host.my-hostname.ru
12      - my-hostname.ru
13      secretName: letsencrypt
14   rules:
15     - host: my-hostname.ru
16       http:
17         paths:
18           - path: /
19             pathType: Prefix
20             backend:
21               service:
22                 name: mipt-kotlin-ktor-load-balancer
23                 port:
24                   number: 8080
25     - host: sub-host.my-hostname.ru
26       http:
27         paths:
28           - path: /
29             pathType: Prefix
30             backend:
31               service:
32                 name: mipt-kotlin-ktor-load-balancer
33                 port:
34                   number: 9090
35
```

✓ 4 ⌂ ⌃

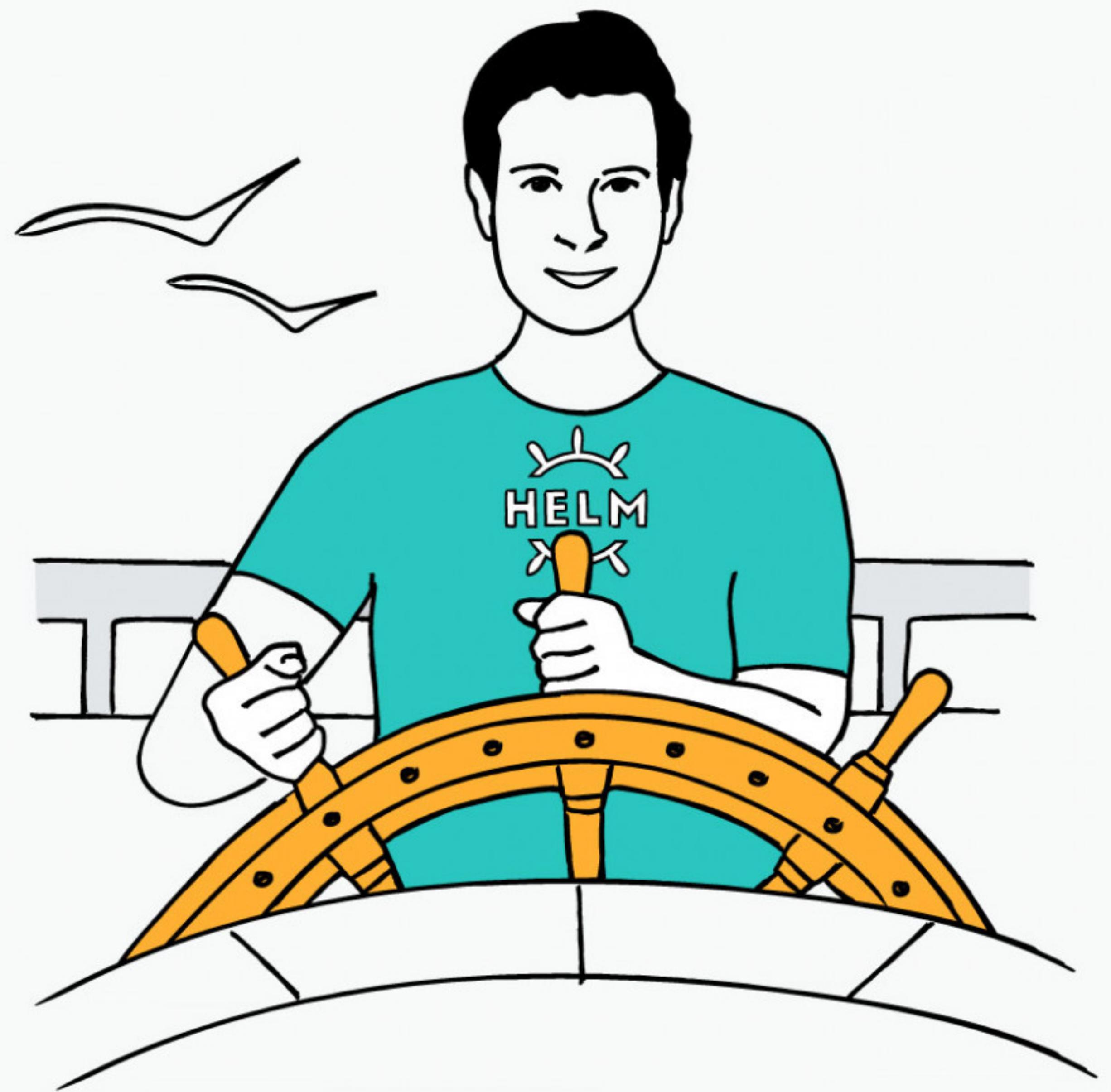
**Что это означает?**

- Все вызовы my-hostname.ru должны попасть в компонент(Service) mipt-kotlin-ktor-load-balancer в порт 8080
- Все вызовы sub-host.my-hostname.ru должны попасть в компонент(Service) mipt-kotlin-ktor-load-balancer в порт 9090

**А теперь давайте запустим k8s  
локально!**

**Какие есть еще технологии  
рядом с k8s?**

# Helm Package Manager

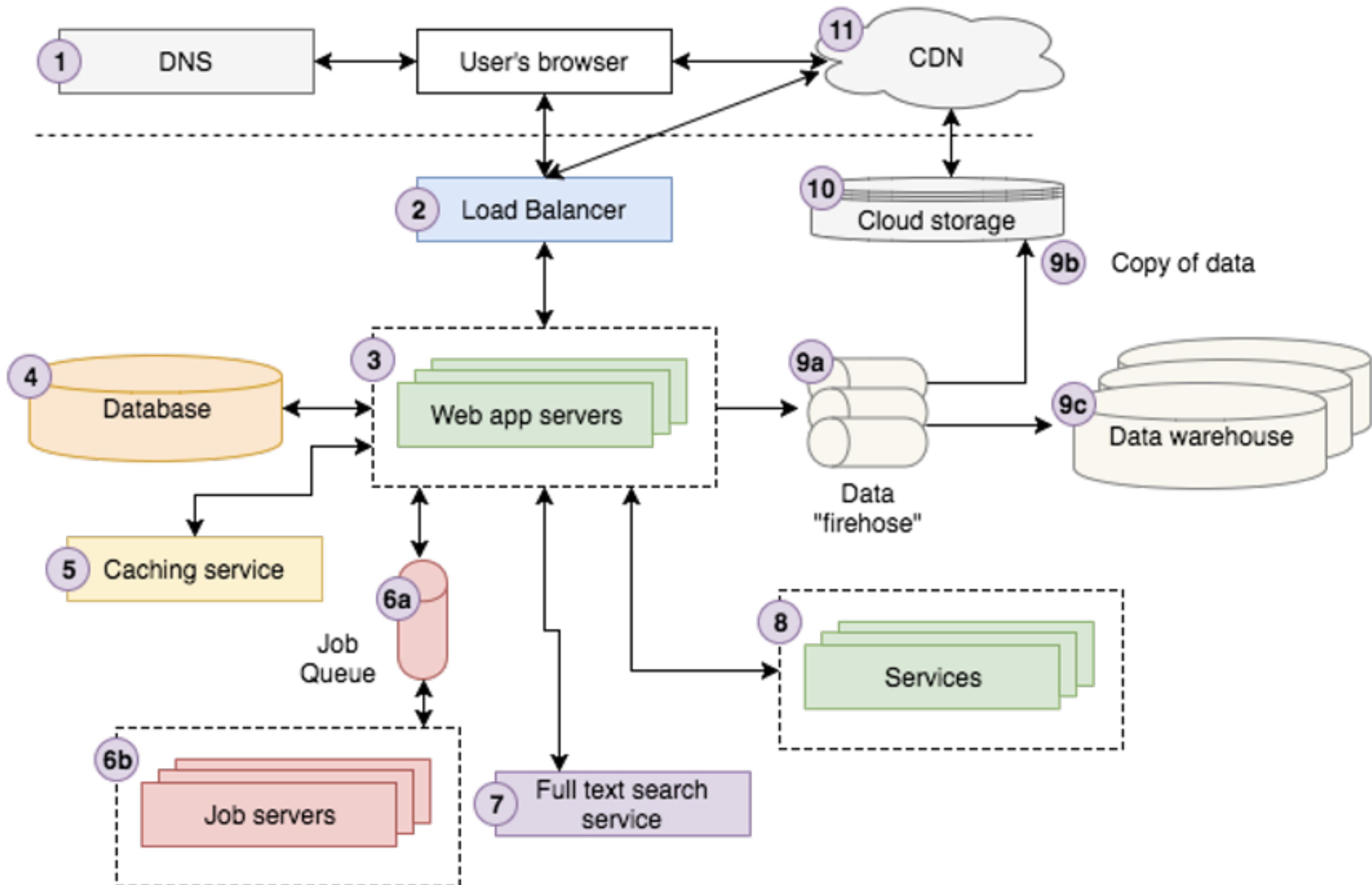


**Что это?**

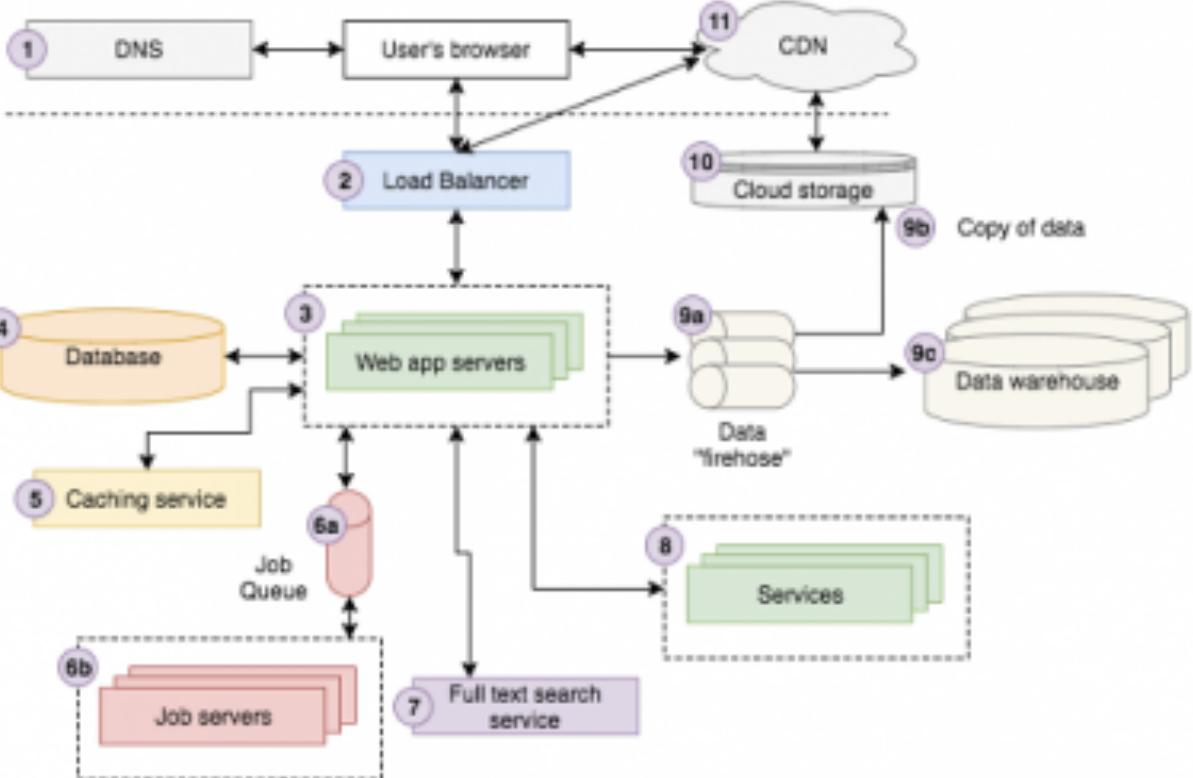
**Один из самых популярных пакетных  
менеджеров для Kubernetes**

# Назначение Helm

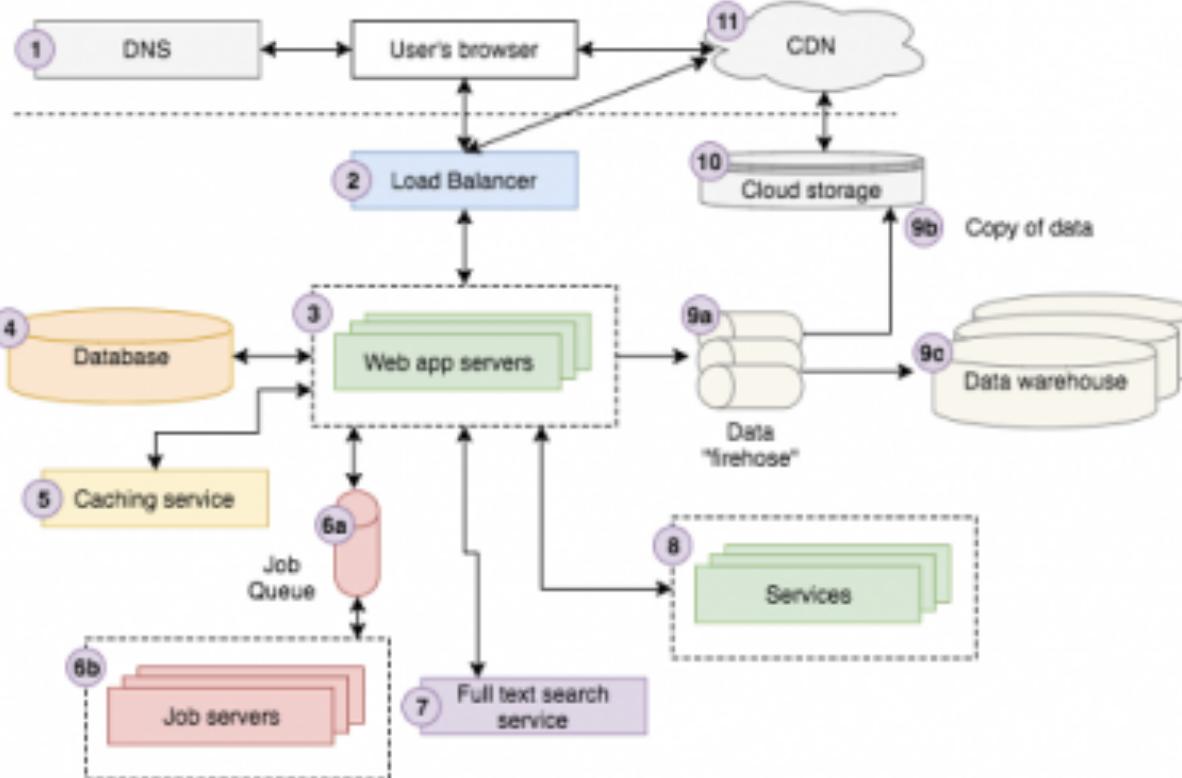
**Посмотрим на современное  
приложение**



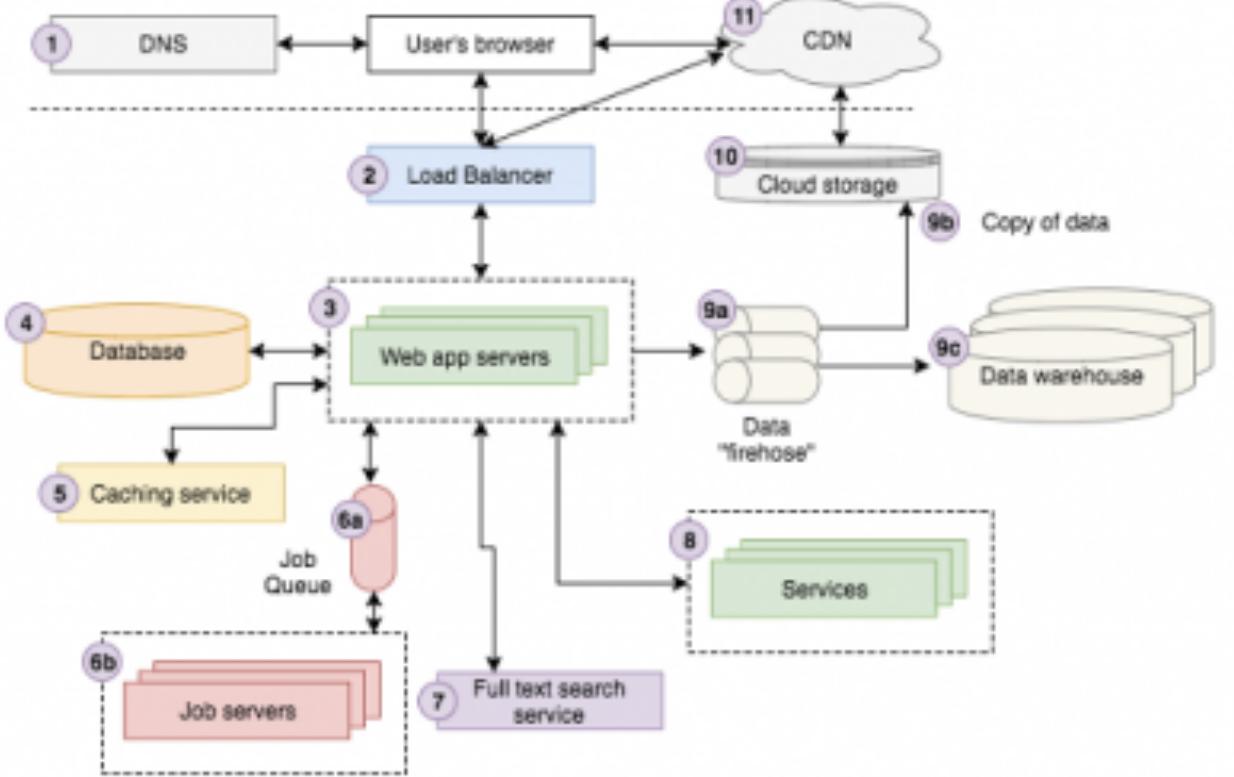
**Но на самом деле, оно  
выглядит так:**



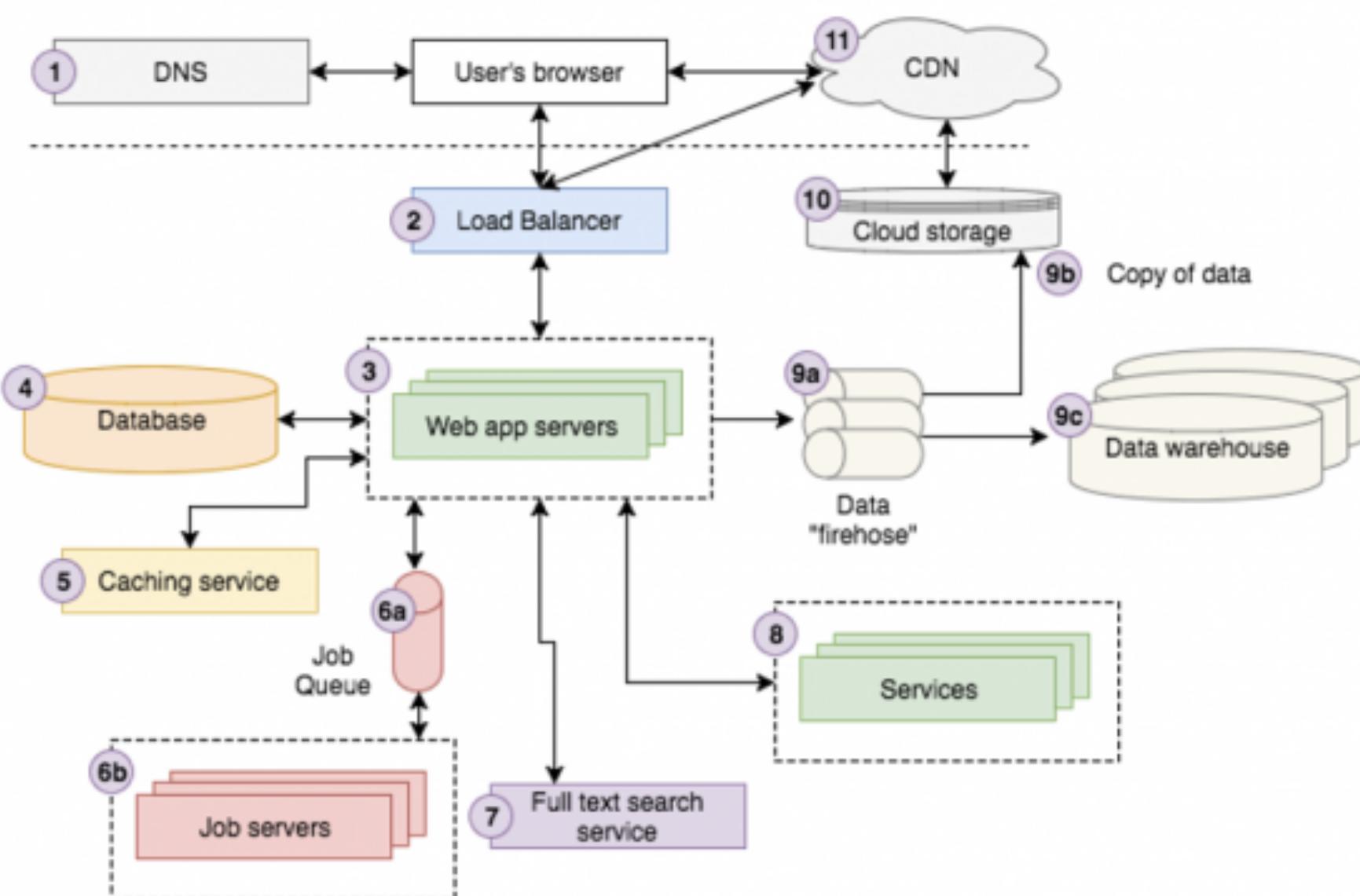
# Dev



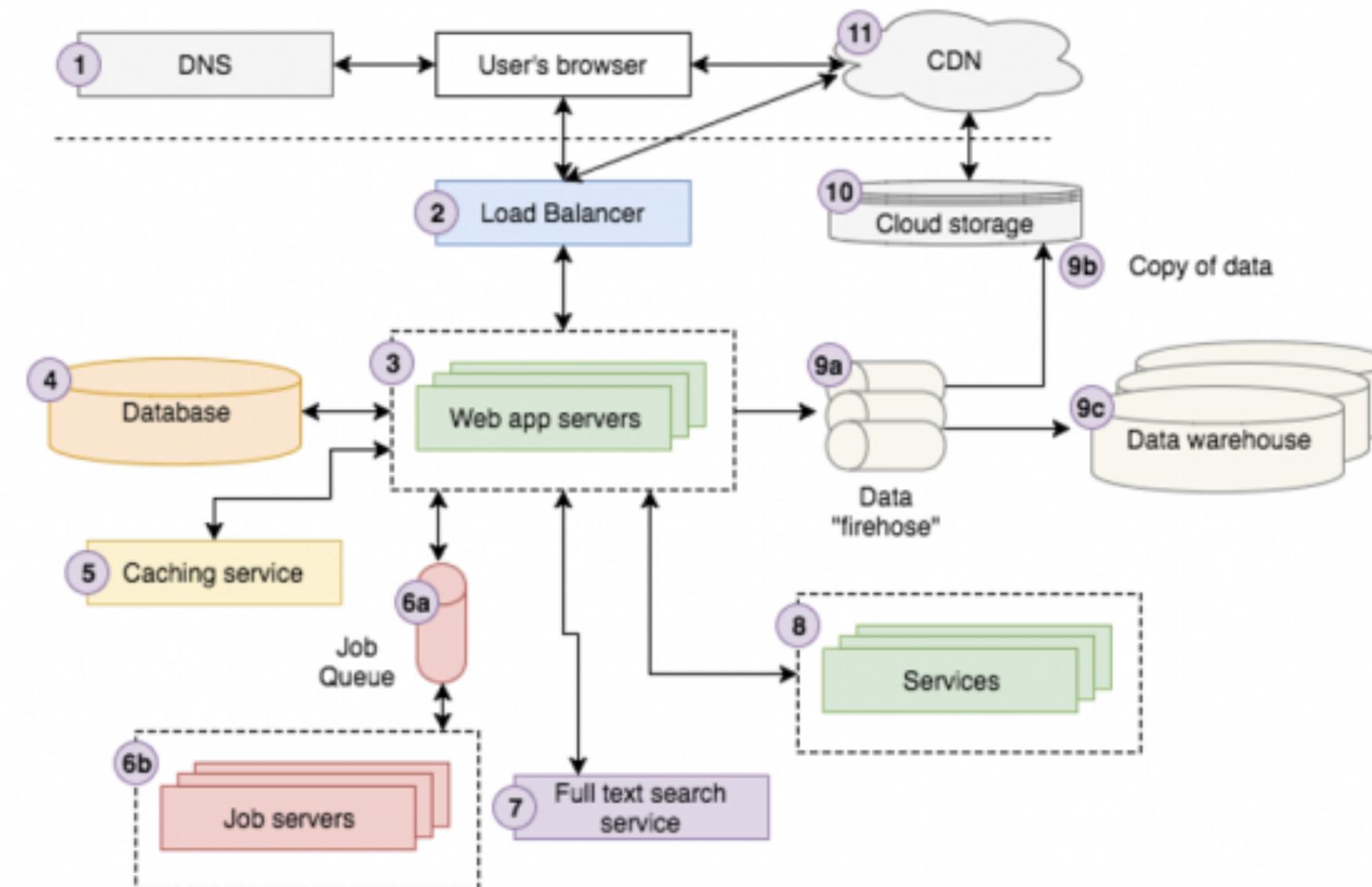
# Test



# Stage

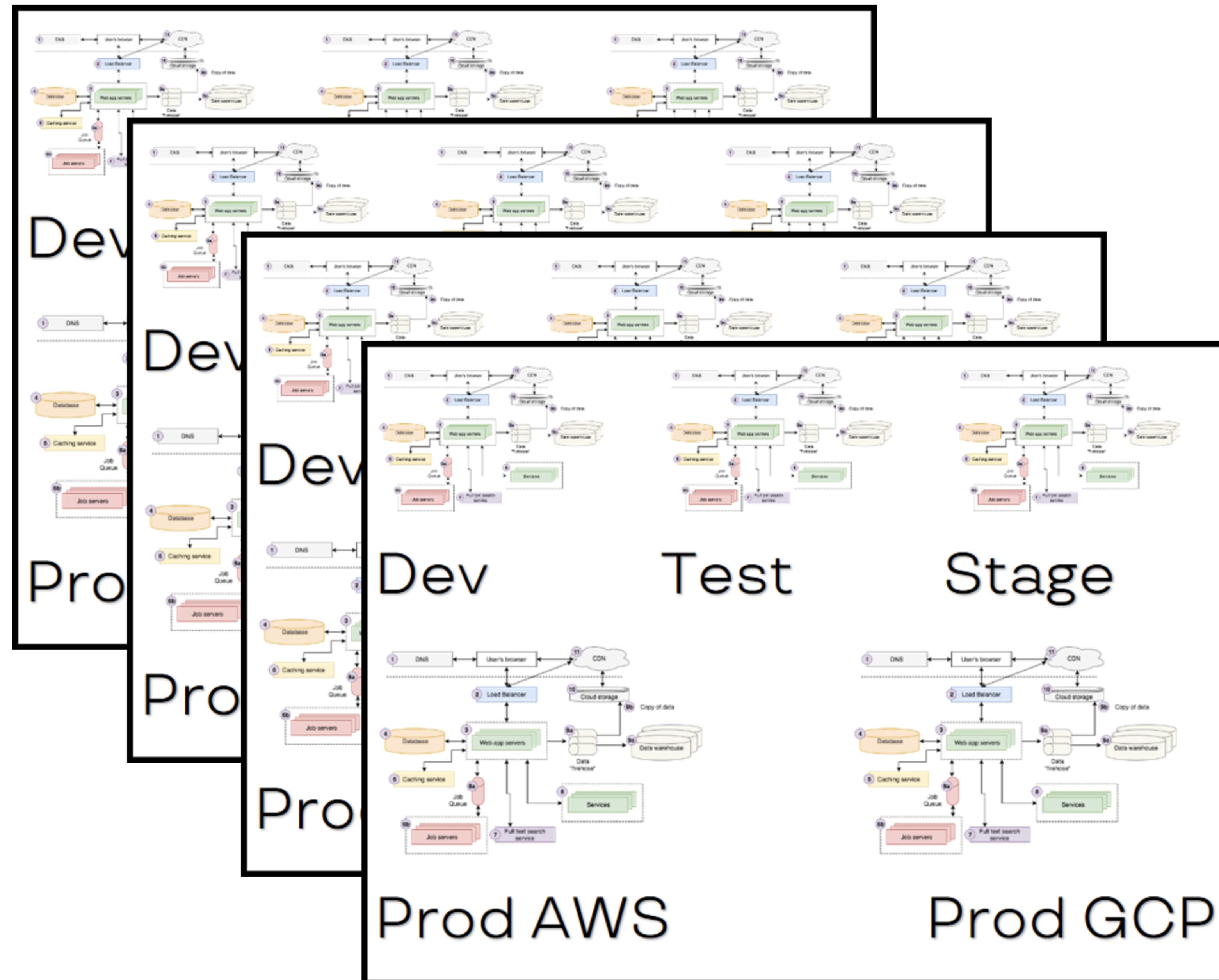


# Prod AWS



# Prod GCP

**А спустя время, оно вообще  
выглядит так:**

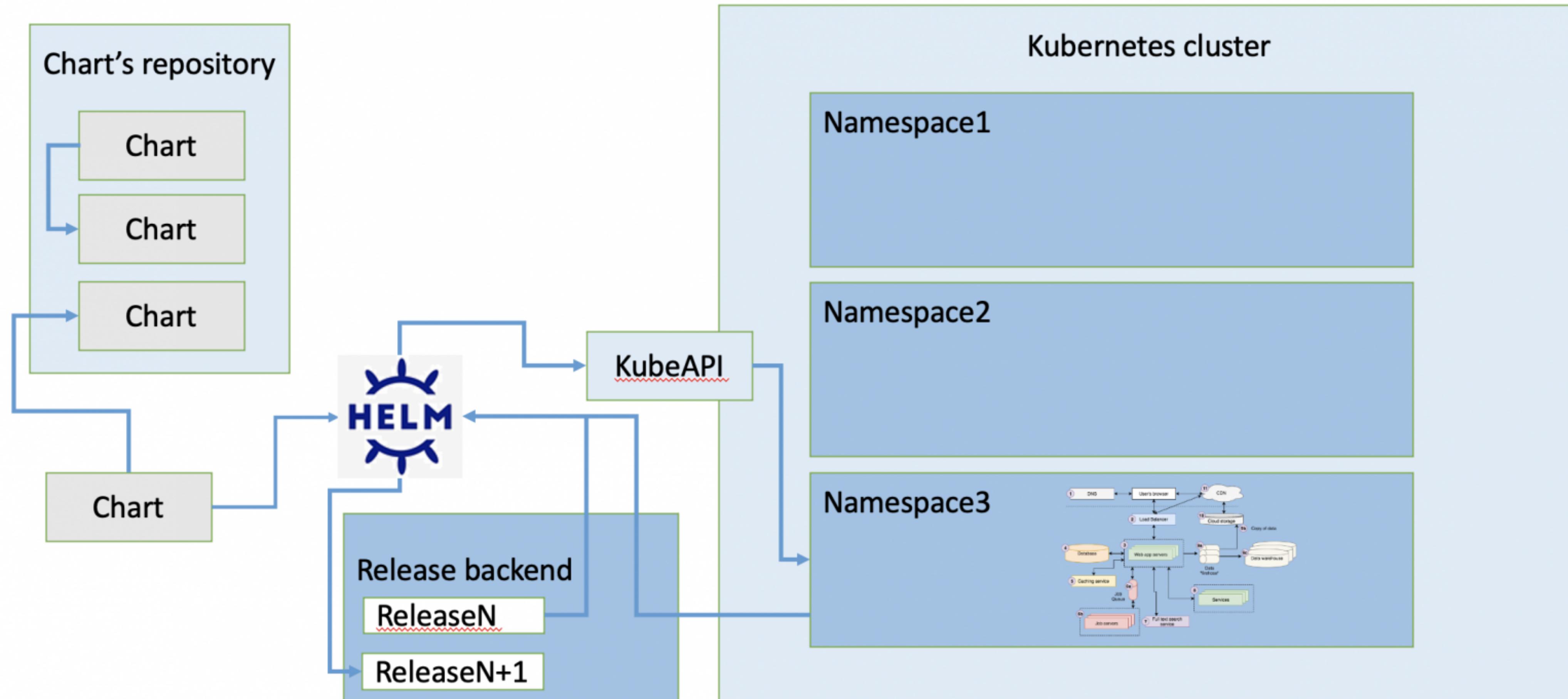


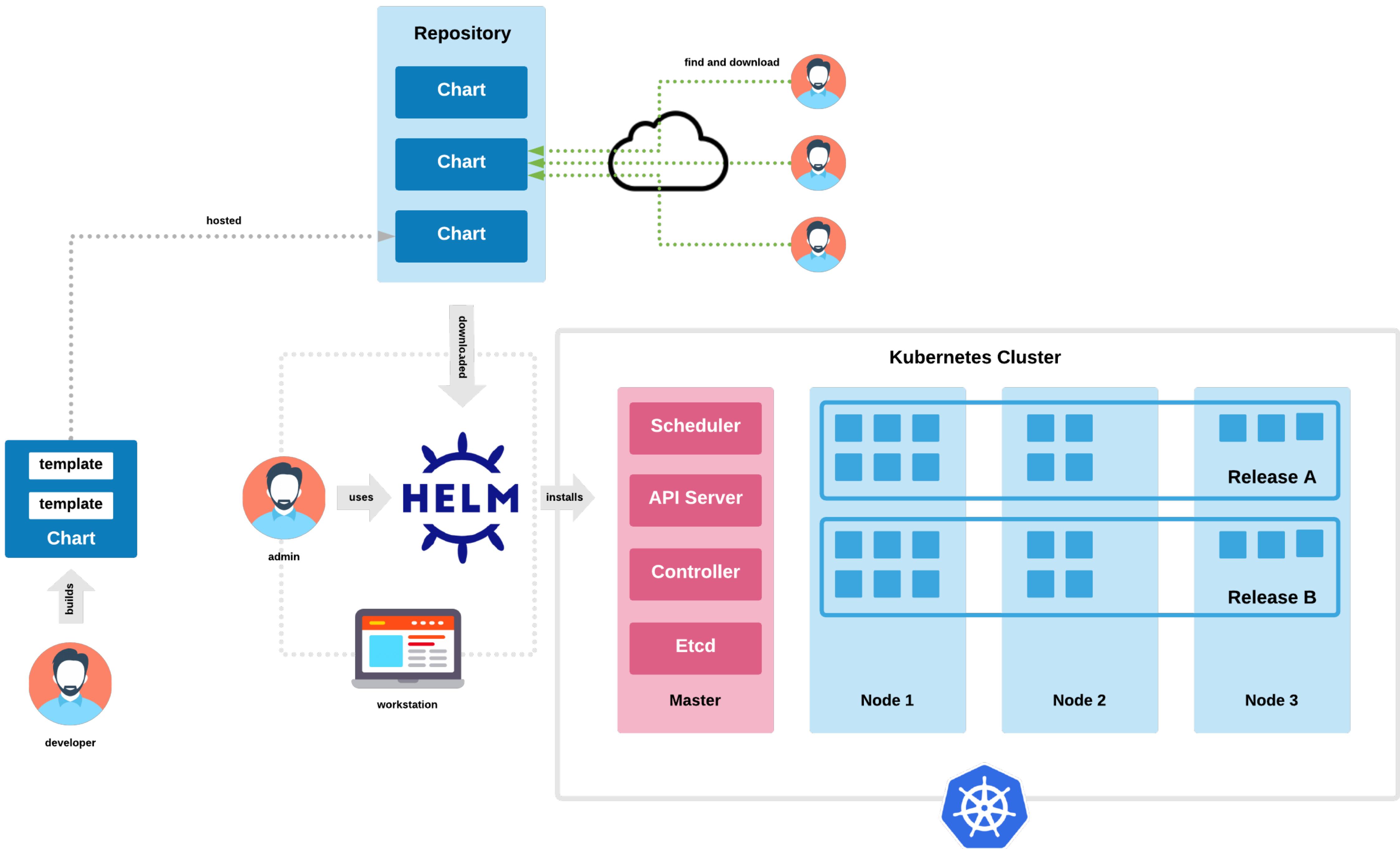
# Project N

**Что хотелось бы?**

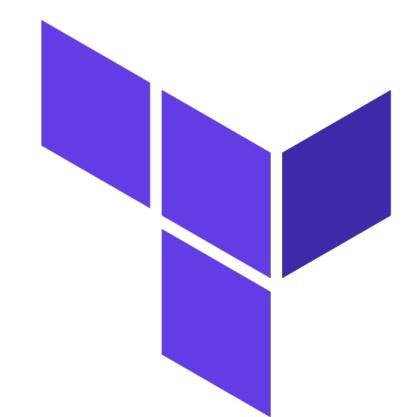
- Управлять этим хозяйством максимально просто
- Управлять универсально
- Не писать каждый раз почти одинаковые гигантские manifest файлы
- Хранить manifest файлы

# **Как работает HELM?**





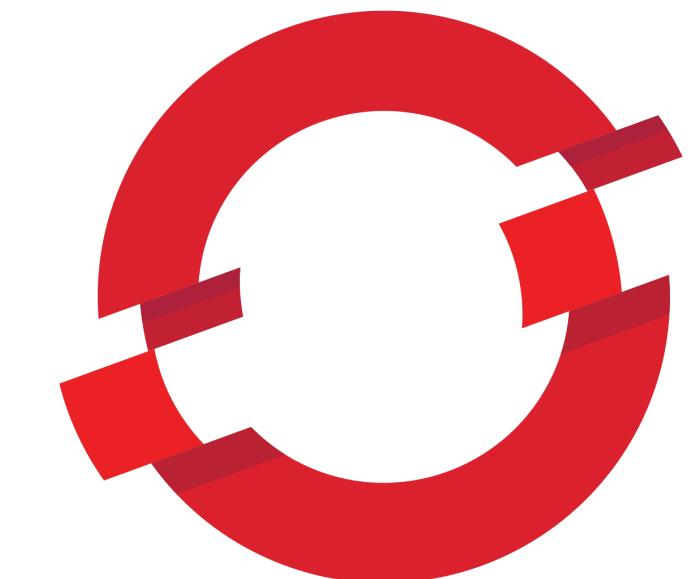
**На самом деле инструментов  
очень много**



HashiCorp  
**Terraform**



**ROOK**



**OPENSIFT**



**HARBOR**



**werf**

# **Итоги**

- K8s очень больший и мощный инструмент
- При этом начать его использовать можно просто и быстро
- Во круг k8s на самом деле есть очень много инструментов
- Изучить все досконально нереально

**Вопросы?**

# Материалы

**Будут в README**

**Спасибо за внимание!**