

# K8s теория

- в целом про объекты в кубере
- Сеть в кубере
- Размещение подов

# в целом про объекты в кубере

Тут я тыкался в чат гпт чтобы разобраться в кубере.

## что такое сервисы в кubernetes?

Сервисы (Services) в Kubernetes - это абстракция, которая позволяет управлять доступом к набору подов (Pods) в приложении или сервисе, предоставляя им стабильное сетевое имя и IP-адрес. Сервисы являются важной частью сетевой инфраструктуры Kubernetes и облегчают взаимодействие между различными компонентами вашего приложения.

Основные типы сервисов в Kubernetes:

1. **ClusterIP**: Этот тип сервиса создает внутренний IP-адрес, который доступен только внутри кластера Kubernetes. Он позволяет подам внутри кластера общаться друг с другом по этому IP-адресу. Этот тип сервиса часто используется для внутренней коммуникации между компонентами приложения.
2. **NodePort**: Сервис типа NodePort открывает порт на каждом узле кластера Kubernetes и перенаправляет трафик на соответствующий сервис. Это позволяет внешним клиентам (извне кластера) обращаться к сервису по IP-адресу узла и порту NodePort.
3. **LoadBalancer**: Тип сервиса LoadBalancer позволяет создать балансировщик нагрузки (Load Balancer) на облачном провайдере (например, AWS, GCP, Azure и т. д.), который затем распределяет трафик на сервисы вашего кластера. Это полезно, если вы хотите обеспечить балансировку нагрузки и высокую доступность для вашего приложения.
4. **ExternalName**: Этот тип сервиса предоставляет внешний DNS-именованный доступ к внешним службам, не связанным с Kubernetes. Это может быть полезно, например, для доступа к внешним базам данных или другим службам.

Сервисы в Kubernetes предоставляют абстракцию для сетевой конфигурации, что делает их более устойчивыми к изменениям внутренней структуры приложения. Они также играют ключевую роль в обеспечении масштабируемости и доступности приложений в Kubernetes-

кластерах.

# т.е. services это просто предоставление сетевого доступа к deployment или pod'ам?

Да, вы правильно поняли. Сервисы (Services) в Kubernetes предоставляют сетевой доступ к подам (Pods) или, более точно, к набору подов, объединенных по определенным критериям (например, селекторам). Это позволяет другим компонентам вашего приложения (или внешним клиентам) обращаться к этим подам, используя стабильное сетевое имя и IP-адрес, независимо от того, где и как они развернуты в вашем кластере.

Сервисы обеспечивают абстракцию от конкретных экземпляров подов и позволяют управлять сетевой конфигурацией приложения, что делает его более устойчивым к изменениям и масштабируемым. Они также позволяют реализовать разные типы балансировки нагрузки и организовать внешний доступ к вашему приложению, если это необходимо.

# разница между деплойментом и репликасетом

ChatGPT

Конечно, я могу объяснить разницу между деплойментами (Deployments) и репликасетами (ReplicaSets) в Kubernetes.

## 1. Репликасет (ReplicaSet):

- **Цель:** Репликасет служит для обеспечения желаемого количества рабочих копий (подов) для приложения в кластере Kubernetes. Он создает и управляет репликами подов.
- **Обновление и масштабирование:** Репликасеты позволяют масштабировать количество реплик (подов) вверх или вниз в соответствии с заданным количеством реплик. Однако они не обеспечивают механизм автоматического обновления приложения при изменении образа контейнера.
- **Селектор:** ReplicaSet использует селекторы для определения, какие поды включать в свою репликацию. Если поды соответствуют селектору ReplicaSet, он будет следить за их количеством и поддерживать его на заданном уровне.

## 2. Деплоймент (Deployment):

- **Цель:** Деплойменты предоставляют более высокоуровневый уровень абстракции над ReplicaSets. Они используются для управления обновлением приложения и его версий.
- **Обновление и масштабирование:** Деплойменты позволяют легко обновлять версии приложения, контролируя количество старых и новых реплик во время процесса обновления. Они также поддерживают откат на предыдущую версию в случае проблем с новой версией.
- **Репликации:** Деплойменты фактически создают и управляют ReplicaSets внутри себя. Это позволяет им управлять репликацией подов, чтобы обеспечить безопасное обновление.
- **Описание:** Деплойменты могут быть легко описаны с помощью YAML-файлов, что делает их более удобными для развертывания и обновления приложений.

Таким образом, основное отличие между ReplicaSets и Deployments заключается в том, что ReplicaSet управляет репликациями подов, в то время как Deployment обеспечивает управление процессом обновления и развертывания приложений, используя ReplicaSets под капотом. Deployment является более предпочтительным выбором, если вам нужно обновлять или разворачивать новые версии приложения в Kubernetes.

# summary

**под:** минимальная единица, один или более контейнеров, внутри обязательно `/pause`, который обеспечивает сетевые настройки

**репликасет:** сведения о реплике подов, больше там ничего быть не может. Конфигмапы, сервисы, секреты не входят явно в репликасет, они управляются другими объектами в кubernetes

**деплоймент:** условно, описание приложения, а именно описание репликасетов и прочих объектов, там указываются репликасеты, с помощью деплоймента можно плавно выкатить, при ошибках автоматически откатить репликасеты. В деплойменте указываются:

- Реплики
- Шаблон пода
- Стратегия обновления
- Стратегия отката
- автоскейлинг
- Контроль обновления (по сколько подов обновлять деплоймент)
- метки, аннотации

абстракцией выше является:

- StatefulSet - управление приложениями с хранением состояния, например мускуль или банально сервак майнкрафта, когда мы не можем тупо размножить приложение, когда у нас домашние животные, а не стадо
- DaemonSet - развертывание одной копии пода на каждом узле кластера, например агент мониторинга, сборщик логов

## сеть в кубере

Сеть в Kubernetes играет ключевую роль, так как она обеспечивает связь между всеми компонентами приложения, работающими в кластере. Организация сети в Kubernetes включает в себя несколько аспектов:

1. **Поды (Pods):** Поды - это минимальные рабочие единицы в Kubernetes, и они могут иметь свой собственный IP-адрес. Поды в одной ноде могут общаться друг с другом через локальную сеть на этой ноде без какой-либо дополнительной конфигурации. Однако для обмена данными между подами, размещенными на разных нодах, требуется сетевой роутинг.
2. **Сетевые плагины (CNI):** Kubernetes использует сетевые плагины, такие как Calico, Flannel, Cilium, и другие, для управления сетью в кластере. Эти плагины обеспечивают создание виртуальной сети, настройку маршрутизации, а также возможность применения политик безопасности сети.
3. **Сервисы (Services):** Сервисы в Kubernetes предоставляют стабильное именование и IP-адреса для доступа к набору подов. Когда вы создаете сервис, Kubernetes автоматически настраивает прокси и балансировщики нагрузки для перенаправления трафика к подам, соответствующим этому сервису. Это обеспечивает сетевую доступность и управление трафиком между подами.
4. **Ингрес-контроллеры (Ingress Controllers):** Ингрес-контроллеры позволяют управлять входящим сетевым трафиком в кластере. Они предоставляют функциональность маршрутизации трафика на основе правил и настройки SSL-терминации.

- nginx
- traefik
- haproxy ingress

5. **Политики сетевой безопасности (Network Policies):** Политики сетевой безопасности позволяют ограничивать сетевой доступ между подами на основе правил. Это помогает обеспечивать безопасность приложений и сетевые политики в кластере.
6. **Сетевая политика (Network Policy):** Сетевая политика Kubernetes определяет правила сетевого доступа для селекторов подов, контролируя, какие поды могут общаться между собой и какие порты и протоколы разрешены.
7. **Интеграция с внешними сетями:** Kubernetes также позволяет интегрировать внешние сети и сервисы, что важно для обеспечения связности приложений с внешними ресурсами и сервисами.

Организация сети в Kubernetes может быть сложной задачей, особенно в больших кластерах с множеством микросервисов. Выбор сетевого плагина, настройка сетевых политик и контроля доступа, а также управление трафиком - это важные аспекты, которые требуют внимания при работе с Kubernetes.

## сетевые плагины

Сетевые плагины (CNI): Kubernetes использует сетевые плагины, такие как Calico, Flannel, Cilium, и другие, для управления сетью в кластере. Эти плагины обеспечивают создание виртуальной сети, настройку маршрутизации, а также возможность применения политик безопасности сети.

В Kubernetes существует несколько различных сетевых плагинов (CNI - Container Network Interface), каждый из которых предоставляет свои особенности и возможности. Выбор сетевого плагина зависит от требований вашего кластера и приложений. Вот некоторые из наиболее распространенных сетевых плагинов и их основные различия:

### 1. Calico:

- **Особенности:** Calico предоставляет богатый набор функций для сетевой политики и безопасности. Он также поддерживает сетевую сегментацию, что полезно для мультитенантных кластеров.
- **Модель маршрутизации:** Calico использует маршрутизацию на уровне узла (node-based routing) и BGP для маршрутизации пакетов между узлами.
- **Применение:** Хорошо подходит для крупных и мультитенантных кластеров с требованиями к безопасности.

### 2. Flannel:

- **Особенности:** Flannel обеспечивает простую и быструю настройку сети. Он может использовать различные механизмы маршрутизации, включая сети на основе оверлея и маршрутизацию на основе хоста.
- **Модель маршрутизации:** В зависимости от конфигурации, Flannel может использовать VXLAN, UDP, Host-GW и другие методы маршрутизации.
- **Применение:** Подходит для простых и средних кластеров без сложных сетевых политик.

### 3. Cilium:

- **Особенности:** Cilium предоставляет сетевую политику и безопасность, а также возможности обнаружения и балансировки нагрузки на прикладном уровне (L7).
- **Модель маршрутизации:** Cilium использует eBPF (extended Berkeley Packet Filter) для управления трафиком на уровне ядра, что обеспечивает высокую производительность и гибкость.
- **Применение:** Подходит для средних и крупных кластеров, особенно для микросервисных архитектур и обеспечения безопасности.

### 4. Weave:

- **Особенности:** Weave предоставляет сетевую сегментацию, маршрутизацию и возможность настройки сетевых политик.
- **Модель маршрутизации:** Weave использует маршрутизацию на основе оверлея (overlay routing) и умеет автоматически обнаруживать узлы в сети.
- **Применение:** Подходит для кластеров, требующих простой настройки сети и сетевой сегментации.

### 5. Kube-router:

- **Особенности:** Kube-router предоставляет сетевую политику, маршрутизацию на основе хоста и дополнительные функции безопасности.
- **Модель маршрутизации:** Использует маршрутизацию на основе хоста и BGP для маршрутизации пакетов между узлами.
- **Применение:** Подходит для кластеров, где важна простота и безопасность сети.

Выбор сетевого плагина зависит от ваших требований по производительности, безопасности, масштабируемости и функциональности. При проектировании и настройке сети в Kubernetes важно учитывать конкретные потребности вашего приложения и кластера.

## Сервисы (Services)

в Kubernetes - это абстракция, которая позволяет управлять доступом к набору подов (Pods) в приложении или сервисе, предоставляя им стабильное сетевое имя и IP-адрес. Сервисы являются важной частью сетевой инфраструктуры Kubernetes и облегчают взаимодействие между различными компонентами вашего приложения.

Основные типы сервисов в Kubernetes:

1. **ClusterIP:** Этот тип сервиса создает внутренний IP-адрес, который доступен только внутри кластера Kubernetes. Он позволяет подам внутри кластера общаться друг с другом по этому IP-адресу. Этот тип сервиса часто используется для внутренней коммуникации между компонентами приложения.

2. **NodePort:** Сервис типа NodePort открывает порт на каждом узле кластера Kubernetes и перенаправляет трафик на соответствующий сервис. Это позволяет внешним клиентам (извне кластера) обращаться к сервису по IP-адресу узла и порту NodePort.
3. **LoadBalancer:** Тип сервиса LoadBalancer позволяет создать балансировщик нагрузки (Load Balancer) на облачном провайдере (например, AWS, GCP, Azure и т. д.), который затем распределяет трафик на сервисы вашего кластера. Это полезно, если вы хотите обеспечить балансировку нагрузки и высокую доступность для вашего приложения.
4. **ExternalName:** Этот тип сервиса предоставляет внешний DNS-именованный доступ к внешним службам, не связанным с Kubernetes. Это может быть полезно, например, для доступа к внешним базам данных или другим службам.

## всякие разные объекты

В Kubernetes существует множество других объектов и ресурсов, помимо ReplicaSets, Pods и Deployments, которые позволяют управлять различными аспектами вашего приложения и инфраструктуры. Вот несколько из них:

1. **!! Службы (Services):** Службы предоставляют абстракцию для доступа к группе подов по уникальному имени и IP-адресу. Они позволяют управлять трафиком, балансировать нагрузку и обеспечивать устойчивость к изменениям конфигурации подов.
2. **!! StatefulSets:** StatefulSets используются для управления приложениями с состоянием, такими как базы данных, где каждый под имеет уникальное и постоянное имя и хранит состояние. Они обеспечивают порядок создания и масштабирования подов.
3. **!! DaemonSets:** DaemonSets используются для развертывания одной копии пода на каждом узле кластера. Это полезно для приложений, которые должны работать на каждом узле, таких как агенты мониторинга.
4. **!! Job и CronJob:** Job используется для выполнения задачи в кластере, которая завершается после выполнения. CronJob позволяет создавать периодически выполняющиеся задачи, аналогичные Cron-задачам в ОС.
5. **!! ConfigMap и Secret:** ConfigMap используется для хранения конфигурационных данных в виде ключ-значение, а Secret - для хранения чувствительной информации, такой как пароли и ключи.
6. **Horizontal Pod Autoscaler (HPA):** HPA автоматически масштабирует количество реплик подов на основе метрик использования ресурсов, чтобы обеспечить требуемую производительность и отказоустойчивость.
7. **Namespace:** Namespace используется для организации и изоляции ресурсов и объектов в кластере. Они позволяют создавать виртуальные кластеры внутри одного физического кластера Kubernetes.
8. **!! Persistent Volumes и Persistent Volume Claims:** Эти объекты используются для управления постоянными хранилищами в Kubernetes, позволяя приложениям



работать с данными, сохраняя их состояние даже после перезапуска подов.

**Долгосрочное хранение! Данные БД например** Типы доступа: много читают, один пишет, много пишут

9. **Network Policies:** Network Policies определяют правила сетевой безопасности и управляют трафиком между подами на основе меток и селекторов.
10. **Service Account:** Service Account определяют уровень доступа к API-серверу Kubernetes для подов и других объектов. Они используются для управления аутентификацией и авторизацией.
11. **Custom Resource Definitions (CRDs):** CRDs позволяют создавать собственные пользовательские ресурсы и контроллеры, расширяя функциональность Kubernetes для конкретных потребностей приложений.
12. **!! Volume:** Volume используется для монтирования директорий и файлов в поды. Различные типы Volume поддерживают разные методы хранения и доступа к данным. **Локальные данные! КЕШ**

Это лишь некоторые из ключевых объектов и ресурсов в Kubernetes. Kubernetes предоставляет богатую экосистему инструментов и API для управления контейнеризованными приложениями и их инфраструктурой в кластере.

## масштабирование

- **горизонтальное** - увеличение/уменьшение кол-ва подов в зависимости от нагрузки
- **вертикальное** - управление ресурсами подов

# Сеть в кубере

## Service

указание, в какой порт пода стучаться.

1. **ClusterIP**: Этот тип сервиса создает внутренний IP-адрес, который доступен только внутри кластера Kubernetes. Он позволяет подам внутри кластера общаться друг с другом по этому IP-адресу. Этот тип сервиса часто используется для внутренней коммуникации между компонентами приложения.
2. **NodePort**: Сервис типа NodePort открывает порт на каждом узле кластера Kubernetes и перенаправляет трафик на соответствующий сервис. Это позволяет внешним клиентам (извне кластера) обращаться к сервису по IP-адресу узла и порту NodePort.
3. **LoadBalancer**: Тип сервиса LoadBalancer позволяет создать балансировщик нагрузки (Load Balancer) на облачном провайдере (например, AWS, GCP, Azure и т. д.), который затем распределяет трафик на сервисы вашего кластера. Это полезно, если вы хотите обеспечить балансировку нагрузки и высокую доступность для вашего приложения. **НА SELFHOSTED КЛАСТЕРЕ ЕГО НЕТ**
4. **ExternalName**: Этот тип сервиса предоставляет внешний DNS-именованный доступ к внешним службам, не связанным с Kubernetes. Это может быть полезно, например, для доступа к внешним базам данных или другим службам.

Скорее всего тебе придется иметь дело либо с NodePort, либо с ClusterIP.

**TLDR:** ClusterIP - айпишник внутри кластера. Это здравый подход, на этот айпишник надо натравлять ингресс. NodePort - проброс порта с пода напрямую на адрес ноды. Не делай так, это некруто.

## Ingress

Описание того, что куда и как надо проксировать. Не путать с описанием ингресс контроллера! Ингресс ничего никуда не проксирует!

## Ingress Controller

Само приложение, которое проксирует условные http запросы на под. Например nginx.

# Простенькое приложение на поиграться с сетями

```
helm repo add hello https://www.kleinloog.ch/hello-helm/  
helm install my-hello hello/hello --version 0.4.0-rc2
```

# Размещение подов

Допустим мы хотим разместить ингрессы на машинах, которые мы (в своей голове) выделили под ингрессы. Одного желания тут недостаточно, один из способов - лейблы на нодах, nodeSelector в манифесте, taint'ы, Toleration'ы.

## Маркируем ноды

Мы можем пометить ноду, чтобы на ней размещались только определенные поды. Делается это так:

```
kubectl label nodes k8s-ingress-1 nodeType=ingress
```

Теперь если мы посмотрим конфиг ноды, увидим там этот лейбл:

```
kubectl get no -o yaml k8s-ingress-1
apiVersion: v1
kind: Node
metadata:
  annotations:
    ...
  creationTimestamp: "2023-10-04T14:44:13Z"
  finalizers:
    - wrangler.cattle.io/node
    - wrangler.cattle.io/managed-etcd-controller
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: rke2
    beta.kubernetes.io/os: linux
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: k8s-ingress-1
    kubernetes.io/os: linux
    node.kubernetes.io/instance-type: rke2
    nodeType: ingress # <--- Вот наш лейбл
```

Мы сделали так, чтобы наши поды попадали на эту ноду и только на нее, но на нее будут попадать также **другие поды**.

Чтобы этого избежать нам нужны **taint'ы**. Теинты указывают, где размещать поды нельзя. Чтобы сказать шедулеру, чтоб он убрал все поды кроме уже запущенных, повесим теинт NoSchedule

```
kubectl taint nodes k8s-ingress-1 ingress-taint=true:NoSchedule
```

```
kubectl get no -o yaml k8s-ingress-1
apiVersion: v1
kind: Node
metadata:
  annotations:
    ...
  creationTimestamp: "2023-10-04T14:44:13Z"
  finalizers:
    - wrangler.cattle.io/node
    - wrangler.cattle.io/managed-etcd-controller
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: rke2
    beta.kubernetes.io/os: linux
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: k8s-ingress-1
    kubernetes.io/os: linux
    node.kubernetes.io/instance-type: rke2
    nodeType: ingress
  name: k8s-ingress-1
  resourceVersion: "1765082"
  uid: XXX
spec:
  podCIDR: X.X.X.X/24
  podCIDRs:
    - X.X.X.X/24
  providerID: rke2://k8s-ingress-1
  taints:
    - effect: NoSchedule # <----
    key: ingress-taint
    value: "true"
```

## Правим манифест подов

Указываем в манифесте пода/хельм чарта следующее:

```
...
nodeSelector:
  nodeType: ingress
...
tolerations:
  - key: "ingress-taint"
    operator: "Exists"
    effect: "NoSchedule"
...
```

Src:

<https://prudnitskiy.pro/post/2021-01-15-k8s-pod-distribution/>