Kubernetes ??? ???????????????

- Обобщенно
- helm
  - Это база
  - Пробуем что-то по серьезнее
  - Пишем свой чарт
- RKE2, helm
- K8s теория
  - вцелом про объекты в кубере
  - Сеть в кубере
  - Размещение подов
- Мониторинг
- Ставим воркер руками
- Создаем пользователей в кубе

?????????

Хороший цикл статей: https://ealebed.github.io/tags/kubernetes/

Видосики: https://www.youtube.com/watch?list=PL8D2P0ruohOA4Y9LQoTttfSgsRwUGWpu6&v=Jp866ltZBSk&embeds_referring_euri=https%3A%2F%2Flulzette.ru%2F&source_ve_path=MjM4NTE

Простой в установке дистрибутив: https://k3s.io/

Делаем

```
curl -sfL https://get.k3s.io | sh -
```

Подключаем дополнение команд

```
k3s completion bash >> ~/.bashrc
source ~/.bashrc
```

????????? ?????-?????? `kubectl get no`

???????

???? ????????? ???? ???????

еще вариант https://github.com/paulbouwer/hello-kubernetes

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

in ~ λ cat ingress.yml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
```

```
  namespace: default
spec:
  rules:
  - host: k3s.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-hello
            port:
              number: 80

in ~ λ kubectl apply -f ingress.yml
```

???????? ???????????

`kubectl run -it --rm --restart=Never busybox --image=busybox:1.28 -- nslookup kubernetes.default`

helm

Пакетный менеджер для K8s

helm

??? ????

Helm это по сути просто пакетник, как в любом дистре, только с возможностью подстановки своих переменных. Под капотом он просто генерит ямлики с деплойментами и прочими ресурсами. Поэтому что в итоге получится - зависит от фантазии автора пакета

?????????

Для работы необходим конфиг к кубу, например по пути `~/.kube/config`. Хельм не надо как-то дополнительно ставить и хранить его данные, он просто подключается к кубу и создает там все что надо. (FIXME). Чтобы его поставить, надо скачать его из официального репозитория, или, как говорят на официальном сайте,

> you can `curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash` if you want to live on the edge

Чтобы что-нибудь поставить, надо добавить репу. Все как в типичном пакетнике в линуксе. Вот, например, добавим репу VictoriaMetrics и официальную репу хельма:

```
helm repo add stable https://charts.helm.sh/stable
helm repo add vm https://victoriametrics.github.io/helm-charts/
```

После чего надо обновить локальный кеш: `helm repo update`

Кроме репозиториев есть еще и хаб, но он выглядит как ~~помойка~~ Dockerhub или AUR (Arch User Repository), который засран кучей чартов и никак толком не модерируется. Чтобы поставить что-то из чарта, необходимо добавить репозиторий, обновить кеш, а потом поставить пакет. Смотри `helm search hub -o yaml bitnami wordpress`

????? ???-?????? ????????

Что мне уже нравится в хельме, так это то что ты можешь получить переменные контейнера (привет, докер) и указать их отдельным файлом при создании. В докере, например, ты должен ручками зайти на докерхаб, посмотреть, какие параметры можно передать в

контейнер и указать их каким-то макаром через ENV файл, докер-композ или еще хрен знает как.

```
# Посмотреть всю инфу о чарте
helm show all stable/mysql
# Посмотреть сам чарт
helm show chart stable/mysql
# Посмотреть переменные чарта
helm show values stable/mysql
```

Вот, например, я поднял мускуль:

```
22:40:32 losted@shit1:~/cube/helm
$ helm install stable/mysql --generate-name
WARNING: This chart is deprecated
NAME: mysql-1680378034
LAST DEPLOYED: Sat Apr  1 22:40:35 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
MySQL can be accessed via port 3306 on the following DNS name from within your cluster:
mysql-1680378034.default.svc.cluster.local

To get your root password run:

    MYSQL_ROOT_PASSWORD=$(kubectl get secret --namespace default mysql-1680378034 -o
jsonpath="{.data.mysql-root-password}" | base64 --decode; echo)

To connect to your database:

1. Run an Ubuntu pod that you can use as a client:

    kubectl run -i --tty ubuntu --image=ubuntu:16.04 --restart=Never -- bash -il

2. Install the mysql client:

    $ apt-get update && apt-get install mysql-client -y

3. Connect using the mysql cli, then provide your password:
```

```
    $ mysql -h mysql-1680378034 -p

To connect to your database directly from outside the K8s cluster:
    MYSQL_HOST=127.0.0.1
    MYSQL_PORT=3306

    # Execute the following command to route the connection:
    kubectl port-forward svc/mysql-1680378034 3306

    mysql -h ${MYSQL_HOST} -P${MYSQL_PORT} -u root -p${MYSQL_ROOT_PASSWORD}
22:40:37 losted@shit1:~/cube/helm
$ helm ls
NAME                    NAMESPACE       REVISION
UPDATED                                 STATUS          CHART           APP VERSION
mysql-1680378034        default         1               2023-04-01 22:40:35.951681832 +0300
MSK deployed        mysql-1.6.9     5.7.30

22:41:50 losted@shit1:~/cube/helm
$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
mysql-1680378034-798b594f7b-kr2dt   0/1     Running   0          76s
```

Выполняем пункты 1-2-3 и проваливаемся в базу в контейнере убунты внутри куба. Либо можно пробросить порт с контейнера на локальную машину, смотри `Execute the following command to route the connection`

???????

```
23:16:27 losted@shit1:~/cube/helm
$ helm ls
NAME                    NAMESPACE       REVISION
UPDATED                                 STATUS          CHART           APP VERSION
mysql-1680378034        default         1               2023-04-01 22:40:35.951681832 +0300
MSK deployed        mysql-1.6.9     5.7.30
23:16:31 losted@shit1:~/cube/helm
$ helm uninstall mysql-1680378034
release "mysql-1680378034" uninstalled
23:16:44 losted@shit1:~/cube/helm
$ kubectl get po
```

```
No resources found in default namespace.
```

??????? ???-?? ?? ??????????

????? ???? ????

RKE2, helm

RKE2

Хочешь применить какой-то параметр? Укажи его на всех control-plane'х, иначе ничего не применится. Например если на уже засетапленном кластере тебе приспичило отключить nginx-ingress, то `disable: rke2-ingress-nginx` надо указывать на всех rke2-server и всех их по очереди перезапускать

Helm

```
helm list -n nginx-ingress -o yaml

# с какими параметрами засетаплен чарт
helm get -n cattle-system values rancher

# пересоздать релиз (например если изменились параметры)
helm upgrade -n nginx-ingress --force my-ingress-nginx nginx-stable/nginx-ingress --version
0.17.1 -f values.yml --set controller.disableIPV6=true
```

K8s ??????

K8s теория

?????? ??? ??????? ? ??????

Тут я тыкался в чат гпт чтобы разобраться в кубере.

??? ????? ??????? ? ???????????

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.

?.?. services ??? ?????? ????????????? ???????? ??????? ? deployment ??? pod' ???

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.

??????? ????? ????????????? ? ??????????

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 кубернетисе

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 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 (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.

??????? (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.

?????? ?????? ???????

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 контейнеризированными приложениями и их инфраструктурой в кластере.

??????????????

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

???? ? ??????

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.

?????????? ?????????? ?? ?????????? ? ??????

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

?????????? ?????

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.

????????? ????

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"
```

?????? ???????? ?????

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/

??????????

Дашборды: https://github.com/dotdc/grafana-dashboards-kubernetes

Prometheus stack: https://github.com/prometheus-community/helm-charts

https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack

?????? ?????? ??????

У меня есть кластер на k0s например, я хочу подцепить к нему какой-нибудь воркер, где угодно. Наброски команд.

По мотивам https://github.com/kelseyhightower/kubernetes-the-hard-way

```
###

root@kube-ctrl:~/add-manually-worker# cat ca.conf
[req]
distinguished_name = req_distinguished_name
prompt             = no
x509_extensions    = ca_x509_extensions

[ca_x509_extensions]
basicConstraints = CA:TRUE
keyUsage         = cRLSign, keyCertSign

[req_distinguished_name]
C   = US
ST  = Washington
L   = Seattle
CN  = CA

# Worker Nodes
#
# Kubernetes uses a [special-purpose authorization
mode](https://kubernetes.io/docs/admin/authorization/node/)
# called Node Authorizer, that specifically authorizes API requests made
# by [Kubelets](https://kubernetes.io/docs/concepts/overview/components/#kubelet).
# In order to be authorized by the Node Authorizer, Kubelets must use a credential
# that identifies them as being in the `system:nodes` group, with a username
# of `system:node:<nodeName>`.

[kube-work-4]
distinguished_name = kube-work-4_distinguished_name
prompt             = no
```

```
req_extensions     = kube-work-4_req_extensions

[kube-work-4_req_extensions]
basicConstraints     = CA:FALSE
extendedKeyUsage     = clientAuth, serverAuth
keyUsage             = critical, digitalSignature, keyEncipherment
nsCertType           = client
nsComment            = "kube-work-4 Certificate"
subjectAltName       = DNS:kube-work-4, IP:10.138.117.7 # IP воркера
subjectKeyIdentifier = hash

[kube-work-4_distinguished_name]
CN = system:node:kube-work-4
O  = system:nodes
C  = US
ST = Washington
L  = Seattle

[default_req_extensions]
basicConstraints     = CA:FALSE
extendedKeyUsage     = clientAuth
keyUsage             = critical, digitalSignature, keyEncipherment
nsCertType           = client
nsComment            = "Admin Client Certificate"
subjectKeyIdentifier = hash




###
for host in kube-work-4; do
  openssl genrsa -out "${host}.key" 4096

  openssl req -new -key "${host}.key" -sha256 \
    -config "ca.conf" -section ${host} \
    -out "${host}.csr"

  openssl x509 -req -days 3653 -in "${host}.csr" \
    -copy_extensions copyall \
    -sha256 -CA "ca.crt" \
    -CAkey "ca.key" \
```

```
    -CAcreateserial \
    -out "${host}.crt"

  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://10.138.117.204:6443 \
    --kubeconfig=${host}.kubeconfig
done

###

root@kube-ctrl:~/add-manually-worker# cat kube-work-4.kubeconfig
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: `cat ca.crt | base64 -w0`
    server: https://10.138.117.204:6443
  name: kubernetes-the-hard-way
contexts:
- context:
    cluster: kubernetes-the-hard-way
    namespace: default
    user: system:node:kube-work-4
  name: default-context
current-context: default-context
kind: Config
preferences: {}
users:
- name: system:node:kube-work-4
  user:
    client-certificate-data: `cat crt | base64 -w0`
    client-key-data: `cat key | base64 -w0`

###

root@kube-ctrl:~/add-manually-worker# kubectl --kubeconfig kube-work-4.kubeconfig get no
NAME          STATUS   ROLES    AGE   VERSION
kube-work-1   Ready    <none>   85d   v1.30.3+k0s
kube-work-2   Ready    shits    15h   v1.30.6+k0s
```

```
kube-work-3   Ready    <none>   14h   v1.30.5+k0s

###

root@kube-work-4:~# cat down
https://github.com/containernetworking/plugins/releases/download/v1.3.0/cni-plugins-linux-amd64-v1.3.0.tgz
https://storage.googleapis.com/kubernetes-release/release/v1.30.1/bin/linux/amd64/kube-proxy
https://storage.googleapis.com/kubernetes-release/release/v1.30.1/bin/linux/amd64/kubelet

root@kube-work-4:~# wget -q --show-progress   --https-only   --timestamping   -P downloads   -i down
cni-plugins-linux-amd64-v1.3.0.tgz
100%[===================================================================================================>]  43.24M  21.5MB/s    in 2.0s
kube-proxy
100%[===================================================================================================>]  54.91M  17.3MB/s    in 3.2s
kubelet
100%[===================================================================================================>]  95.46M  22.5MB/s    in 5.0s
apt install containerd -y

###

root@kube-work-4:~# systemctl cat kube.service
# /etc/systemd/system/kube.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes
After=containerd.service
Requires=containerd.service

[Service]
ExecStart=/usr/local/bin/kubelet \
  --config=/var/lib/kubelet/kubelet-config.yaml \
  --kubeconfig=/var/lib/kubelet/kubeconfig \
  --register-node=true \
  --v=2
Restart=on-failure
```

```
RestartSec=5

[Install]
WantedBy=multi-user.target

###

root@kube-work-4:~# cat /var/lib/kubelet/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/var/lib/kubelet/ca.crt"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "10.32.0.10"
cgroupDriver: systemd
containerRuntimeEndpoint: "unix:///var/run/containerd/containerd.sock"
podCIDR: "100.64.32.0/24"
resolvConf: "/etc/resolv.conf"
runtimeRequestTimeout: "15m"
tlsCertFile: "/var/lib/kubelet/kubelet.crt"
tlsPrivateKeyFile: "/var/lib/kubelet/kubelet.key"

###
kube-worker-4.{crt,key}->/var/lib/kubelet/kubelet.{crt,key}

###

root@kube-work-4:~# cat /var/lib/kubelet/kubeconfig
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: # ca base64
```

```
    server: https://10.138.117.204:6443
  name: kubernetes-the-hard-way
contexts:
- context:
    cluster: kubernetes-the-hard-way
    namespace: default
    user: system:node:kube-work-4
  name: default-context
current-context: default-context
kind: Config
preferences: {}
users:
- name: system:node:kube-work-4
  user:
    client-certificate-data: # base64
    client-key-data: # base64


###

systemctl start kube.service

root@kube-ctrl:~# kubectl get no -o wide
NAME          STATUS   ROLES    AGE   VERSION       INTERNAL-IP      EXTERNAL-IP   OS-
IMAGE                         KERNEL-VERSION                CONTAINER-RUNTIME
kube-work-1   Ready    <none>   85d   v1.30.3+k0s   10.138.117.150   <none>        Debian
GNU/Linux 12 (bookworm)   6.1.0-23-amd64                containerd://1.7.20
kube-work-2   Ready    shits    16h   v1.30.6+k0s   10.13.37.3       <none>        Rocky Linux
9.4 (Blue Onyx)      5.14.0-427.42.1.el9_4.x86_64   containerd://1.7.22
kube-work-3   Ready    <none>   14h   v1.30.5+k0s   10.138.117.79    <none>        AlmaLinux
9.4 (Seafoam Ocelot)   5.14.0-427.42.1.el9_4.x86_64   containerd://1.7.22
kube-work-4   Ready    <none>   35s   v1.30.1       10.138.117.7     <none>        Ubuntu
20.04.6 LTS              5.4.0-200-generic             containerd://1.7.12
```

??????? ????????????? ? ????

Предположим мы создали куб (on-premise или в облаке) и нам выдали админский кубконфиг, где авторизация происходит по сертификатам, например:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURBRENDQWVpZ0F3SUJBZ0lVTVhZMjhWMEJaQld2Ry9hQnNZdDJ5OV
prRmVRd0RRWUpLb1pJaHZjTkFRRUwKQlFBd0dERVdNQlFHQTFVRUF4TU5hM1ZpWlhKdVpYUmxjeTFqWVRBZUZ3MHlOakEw
TURrd056TTBNREJhRncwegpOakEwTURZd056TTBNREJhTUJneEZqQVVCZ05WQkFNVERXdDFZbVZ5Ym1WMFpYTXRZMkV3Z2
dFaU1BMEdDU3FHClNJYjNEUUVCQVFVQUE0SUJEd0F3Z2dFS0FvSUJBUURYOE9CZW9wN1VyektEbnpjQzRUNFNjZWVKZndG
VFkvZWsKNXZQQjhIWlFWUWhzRzg5dlMrZEdJdkRZK2ZmMjd0YjZCVUs3ajdGd1Jta3ROSFpNSnNuUEVuSEZvdWp6YlkxSQ
o2SUhQVEY2TXE2VVJnU0NGSWtEVWRrd3JTZ0xQSFlKK3Bic0hNSFF2bW81ZFRQV1dCVzlVeVpxSnFzeW80WjBvCjVaS0VH
c01reUFkVFE1b1J4d2Z3MkMwb1YyZ2N2dWxSdWI1aUtCVXBYRTM4ZGkrcEZ6WkNCaEp3NUQ2UU90cXgKUjlydHZtelRmRm
JETlo3WkdEZlM4cFN4RTBwUThQZEF5M2pjQVpacGpueGZvYldueTdYRGVHOFJkQktzMTJMWApFK3RuSE9HRVB3ck5hb1Q1
MTg4R3R3a095dGxvTEgyanlwUTJtT05FMTJxZlMrVzRnR2hsQWdNQkFBR2pRakJBCk1BNEdBMVVkRHdFQi93UUVBd0lCQm
pBUEJnTlZIUk1CQWY4RUJUQURBUUgvTUIwR0ExVWREZ1FXQkJRVGlNbkYKWDhkUWhQYWdzZ2VXRWJNYXpnRDdrVEFOQmdr
cWhraUc5dzBCQVFzRkFBT0NBUUVBYTVjOWl1YUdGNndiUGcwdApBZlRFRXUrbEZXMXlvRU4zWlU4T0lHdjIzMWh3ekRDVV
dmYXlNOTkrcS9hbnlUbU5YMWZ4eExtMGNDMk9lRkJXCnM2bWc2STF1SFJrOTdJRU1FNGZkNFlhNVVoOGR0NTh0Vld3T3JS
S1hoUFFkZWNTVGVISnM2MzBtcktsa1k2aTcKZTZFMHk0UU80cjBRVzN1N1hTMDNZSEpZc0duVCtmYVdSSTBmcEoyWWQ4OT
A4SS9FbzBmMDR6bjlmYjZOVE5wMApwOHJUeDNpallhek1GNk1seHI1OFJoRmcxa1k5dWZEM1NsdkN5Q1F4Q2hDMUZKelBQ
aVNMcTZFNkt2VFloTHVzCkNTbW54WnRtMkh6Z25BWGE5aVRsN3krNDNQZDM3bEZ2VTJIQnRMczY0YjIrLzNlbjlKL2wwWi
twVUJkVzhxenMKNXdVb0xnPT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://10.13.37.199:6443
  name: local
contexts:
- context:
    cluster: local
    user: user
  name: Default
current-context: Default
kind: Config
users:
```

```
- name: user
  user:
    client-certificate-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURXVENDQWtHZ0F3SUJBZ0lVVnlkV283dkZQOHd2N1kwWXJmVGwrY2
tBeEdrd0RRWUpLb1pJaHZjTkFRRUwKQlFBd0dERVdNQlFHQTFVRUF4TU5hM1ZpWlhKdVpYUmxjeTFqWVRBZUZ3MHlOakEw
TURrd056TTBNREJhRncweQpOekEwTURrd056TTBNREJhTURReEZ6QVZCZ05WQkFvVERuTjVjM1JsYlRwdFlYTjBaWEp6TV
Jrd0Z3WURWUVFECkV4QnJkV0psY201bGRHVnpMV0ZrYldsdU1JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThBTUlJ
QkNnS0MKQVFFQXg1MmtpTHJtd1RHcG5YTXd5NStldnR0MzFtMzRBTEhtVGkyOE1xcjN5YTVRWUpEK1hYWEwrSzlZMForNQ
pXSjhESlhUSzJaendvenJIZFZjUXZpNm9iSDNFUUdhdGtSQXdiVkFvek90OGl1bzYzdUNaaU9GRHhRdWpBZ3l6Cm1uOW1t
cTNobi9STHVEWVE1LzlyWFhYZGduYml4SmF4ZXNoQ2U2ZUptSS8rYkNYaDVyWGtxRXlpUFVtck5zcmgKZ2xiL1JpOG5FME
RRaDRsZG9jOHpjNEtJVnFHT1JJQmhtWXMzMmRuWnduSStHYTlWbTRMQXRXdmlPd0RhMk9zZQpqSXBMT1FhSStUNklHUGh5
blpTU1pUVjFYamZ1ZHlIRWxQSUVENFlMeWIrRnY2bjQ5WGN4NDA2UU9veFB6ek9zCkNGc0pxOENvM0s5eUlINnJkcVJtYm
pSYVh3SURBUUFCbzM4d2ZUQU9CZ05WSFE4QkFmOEVCQU1DQmFBd0hRWUQKVlIwbEJCWXdGQVlJS3dZQkJRVUhBd0VHQ0Nz
R0FRVUZCd01DTUF3R0ExVWRFd0VCL3dRQ01BQXdIUVlEVlIwTwpCQllFRk12NnRlejMzalFDTS85UkM3YkMrRGtNNm5JeU
1COEdBMVVkSXdRWU1CYUFGQk9JeWNWZngxQ0U5cUN5CkI1WVJzeHJPQVB1Uk1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFR
Q1UwRnNueVpPcG5OZFY1TEtsRTRFOC8rTXMKdlR6S1FJUkk1RUZMZ0JicXBhUzJmYTlNL3pDUElua1U2Yk53RTZTWE80Ym
tORmkyWjRzSSt2aDJHS1VKWWNlNwpROFk4YmNxTVoxSWI5Smo1TXNkekJCOFV5YmZ1d2hCNFB6WWc5c3l3MHNBWCtWL3Nr
UEt6YlBUNFY1WUdnRFhiCm9ra2orOWo3U05IMHNiTkdwTjJGV2dCRisvNlNZaFJEclpZdlZVWmsybzJNUk9zRXFhcUlhej
NWQ25WUWYzMzkKRVVtV1dDdFg2VER6ejFEVm44M0lYQ2ErcWlVb3dEYVpGNTMwbFRPSHVNb0NHWGdtcnQ3ZnFpR3FzNFFO
T2RVbgpWOUhaSURFQ0xsNXJvRVoxdDA1Smw2aWdrVkt3dUdzZkRtaTNOV0xRL05od1lSSkNSVDUrbEN6UDNweUQKLS0tLS
1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    client-key-data:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBeDUya2lMcm13VEdwblhNd3k1K2V2dH
QzMW0zNEFMSG1UaTI4TXFyM3lhNVFZSkQrClhYWEwrSzlZMForNVdKOERKWFRLMlp6d296ckhkVmNRdmk2b2JIM0VRR2F0
a1JBd2JWQW96T3Q4aXVvNjN1Q1oKaU9GRHhRdWpBZ3l6bW45bW1xM2huL1JMdURZUTUvOXJYWFhkZ25iaXhKYXhlc2hDZT
ZlSm1JLytiQ1hoNXJYawpxRXlpUFVtck5zcmhnbGIvUmk4bkUwRFFoNGxkb2M4emM0S0lWcUdPUklCaG1ZczMyZG5ad25J
K0dhOVZtNExBCnRXdmlPd0RhMk9zZWpJcExPUWFJK1Q2SUdQaHluWlNTWlRWMVhqZnVkeUhFbFBJRUQ0WUx5YitGdjZuND
lYY3gKNDA2UU9veFB6ek9zQ0ZzSnE4Q28zSzl5SUg2cmRxUm1ialJhWHdJREFRQUJBb0lCQUFXQmgvWC9pUGxNb2VKZwpC
WVlKN2NxR2c4K1pDKzE3WVNBdXVDc2JuU2NZNGVRV1pWd295dkUzWjEyczIzMElaUUZTbGVNQldiTVNIa200Ci9TeTg5dT
I3S2ZLN20ra0FoMm82NFlCc3RHdzdUU1NhZDVHeEZjYkNVNE1HM0hhVC90YzZBQUcxKzZSTU8rRlkKdTFxQ0dlQ05FeVFD
Q1VsbjBiR2pxN0tqc2lYMjN3Zld2RkJRZzBLSlR1TnEycVhqYU5YQ2RyU2NqNStLYVBORwpzVGhCU2JVYjVvMDVrcXRsd3
A5U1NoenZzeHBVWFBoTmNCYjVQRlZmSG12L3F3NzB5L0cwQmhoUGVxV050RmJVCkYzL1k5aEJrMVhtZ1NNVmNIVFUrT3ZW
QXNjVm1OVnBvVVhHQ2VHNEE0MWRCK1lWb24zZnE5Vkp1RzFuR0MwU3gKbDhUdXZmRUNnWUVBMGRzMlZwWmtFUHVrR05RdU
NVSjBocGVQV0RsOThSN3lNY1hvNVJoRS9NWGhvR3lGUFFPMQphUkxpeS90MFVJNFFkUjVYSWQ4RTgyQ0lYQ0x5azN0RUN3
bHVBUzhQc2lJVHg4bHRjbERER25ZTlBVWEZzK2kwClQ5SmFOMFNpL2o0Q2lkTjUrYm1VNnA0Q3h1bXNvZzEzTG5zTUhVUi
8rM1pPaXBoazEzaFQxaEVDZ1lFQTg0SCsKVmpIM2VCV1E2NjlWbmRkSFU4dWdvbUwyeFNxekRvOTkxaE1ydFBJNUlVWndO
ODZNWUhYcWtwRS9KcUI0cXp0YwpNaXh4MG9QU2hYM2taYlNpcENjbzl4d09HYU5uTWprNFY0Z0M0eDYwRzJIcFY1THNFUm
EzaU50UjhaUDdhMTBQCnVMQ3JaUTFmNit3bkowd0hzRHR6UFlYaWJVcmxJdlZ2dHBoaytXOENnWUVBbWFXRkdFcE9BMzBx
ZlI5RWdYVU4KZWNtWG93R2M2eUE4TlNMd3pHTkRoMHFlVW1XQis5VXVTanNRb0VaL3Q4YjcxN1FhR1d0KzVXNGxDRWh5RU
hicwpyNlA4elpNV1M5YlZZcTVnbXBUMDgvZkE0NzZrN0g4UkhXd21yMVpxZS9rTXhMcDRFTlhHYVN5OVhjT1NxS2R4ClBp
```

```
L0xBOWEyV1hjYU5ERTFpK2pHZ1BFQ2dZQk8vRm1OaUFPbmxnYytDOCtQdFdiVGpYZDdkUEpxbmlFYWxmelIKVmNLVm5RUm
ZBTVFodGdQZXZpRHFqZWFZRnZGTlNhSHNQSEpuUUk4bThlRUdCSVBGRDFiQVlQOUozYkQ1bjRuaApDcU0xSEo1N1MxVXZM
TjhaNCs2QW0vT0drdU80dmFUOTJZQ2U5S21xa3gxWUo5ZE9tTm9XbUxrTGpvR294MGdIClNJTm9UUUtCZ0Q3SWV3MnBJeT
FzRGg4bUxUbGJQS294ejhCWnZNK2RhMmpOVjN5enp1ZHRSbkNXL0o3MWl4MVIKZU1TbFpXcGRZR09NYWltUU1OMnp2QjdG
UCtPNnpxaE16SldIL0Y2dm92c2lpWExsQmJpcmNTUnBYYVZyOERZdAo2di9XRHFMRlh1b0tmKyttM1FkZzhlaTY0MmZsWG
J0aVBoWVRwdlhLSTl6dU5hdXFLV0lnCi0tLS0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg==
```

Мы радостно начинаем раздавать этот кубконфиг всем сотрудником нашей дружной организации и всем хорошо. Но всем хорошо до того момента как кто-то уволится и заберет с собой этот кубконфиг.

Возникает вопрос - как отозвать у уволившегося сотрудника данный сертификат. Сейчас уже никак, потому что клиентский сертификат выпускается от корневого сертификата кластера и действует 1 год (обычно), а kubernetes не имеет встроенных механизмов отзыва сертификатов.

Поэтому если с кубом работает больше одного человека - лучше заранее научиться управлять пользователями в кубе.

???????? ?????????????

В кубе есть несколько механизмов (тык) для авторизации. То что мы имеем на руках - "X.509 client certificates", о его минусах я писал выше. Из того что (как мне кажется) больше всего подойдет для живых пользователей: "Service account tokens" и "OpenID Connect Tokens". По второму можно найти инструкции в интернете, а по SA Tokens инфы довольно мало, о нем и пойдет речь.

Авторизация по SA токенам удобна тем что не требуется внешний OpenID сервер и аккаунт вместе с токен(ами) можно отозвать в одну команду. Однако лучше всего все таки настроить OpenID сервер, потому что он выдает короткоживущие токены, что безопаснее.

За основу возьмем конфиг выше. Нам надо будет создать SA в кубе, повесить на него ClusterRoleBinding (можно более точечно выдать роли, но ради упрощения выдадим "cluster-admin"), сгенерировать токен и сформировать кубконфиг.

Создаем пользователя, выдаем ему доступы, генерируем временный токен:

```
# Создаем аккаунт (можно указать неймспейс)
[root@k8s-istio ~]# kubectl create serviceaccount ivan
serviceaccount/ivan created
# Вешаем clusterrole на Ивана
[root@k8s-istio ~]# kubectl create clusterrolebinding ivan-access-full-cluster --
clusterrole=cluster-admin --serviceaccount=default:ivan
```

```
clusterrolebinding.rbac.authorization.k8s.io/ivan-access-full-cluster created
# Генерируем токен, например на 720 часов
[root@k8s-istio ~]# TOKEN=$(kubectl create token ivan --duration=720h)
[root@k8s-istio ~]# echo $TOKEN
eyJhbGciOiJSUzI1NiIsImtpZCI6IkdkNmc0RU5lb0ZiT254dWxSQS1yU3B4NGhLMlZIWGVCc3MzYWxMdjRmWHcifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjIiwic3lzdGVtOmtvbm5lY3Rpdml0eS1zZXJ2ZXIiXSwiZXhwIjoxNzc4MzE1NTI0LCJpYXQiOjE3NzU3MjM1MjQsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2YyIsImp0aSI6IjY2OWM0YmJkLTU4NDUtNDBjYy04ZDEyLTdhYWY0ZTY2Y2U1MiIsImt1YmVybmV0ZXMuaW8iOnsibmFtZXNwYWNlIjoiZGVmYXVsdCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJpdmFuIiwidWlkIjoiNGVjOWE1YTUtMjE0MS00M2IzLWE4YjgtYjNmOWE4NzQ3YmJhIn19LCJuYmYiOjE3NzU3MjM1MjQsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0Oml2YW4ifQ.VM9stmAYwvZg-6KDOFd_rD9lqf4RHSh9U-iahsncKEfUHZzwBYpE3tJ3ZxZvJ3G0STln1BKaioeLKn_MiSIY4ttn8zvlK_wQs-fslOsAAw5mnRYDrD7f-9H1c4P_RQo8dRULmYiCHIlq3OqqPpJtIUT1qiPEyIotThGxufQ5MEdmKDUExN9HIeaQbZbOX-lj1kKqETyuhQGrTjVqCs2G7gPSB3vPbgIAB71PxnvChwsK5M3Ijd1VarKN3rq87g-Qpo-xXLyvNfXyr4_RjUw9_4HokC7HhlOUI_NXYYN0lVzoKJeYVz8CJeIqGAv5CLmlnhvask5rbTGwau7VZStLpw
```

Выпускать токен без срока годности можно через создание секрета - так можно будет точечно его отозвать:

```
apiVersion: v1
kind: Secret
metadata:
  name: ivan-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: "ivan"
type: kubernetes.io/service-account-token
```

```
# Создаем секрет - токен будет создан кубернетисом автоматически
[root@k8s-istio ~]# kubectl apply -f secrey.yaml
secret/ivan-token created

# Получаем токен (может появиться не сразу)
[root@k8s-istio ~]# kubectl get secret ivan-token -o jsonpath={.data.token} | base64 -d
eyJhbGciOiJSUzI1NiIsImtpZCI6IkdkNmc0RU5lb0ZiT254dWxSQS1yU3B4NGhLMlZIWGVCc3MzYWxMdjRmWHcifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6Iml2YW4tdG9rZW4iLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiaXZhbiIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjJiNWNlOWUxLWY0ZWYtNDVkZS1hMmZmLTNjNjRlNDdjZDNmZiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0Oml2YW4ifQ.NE-kMaUoGK-
```

```
2zvzk1bGUF52kwohavmGOHw3SCfG7Leh52mbfB8HmeJkCMcZKO28DREvh9GiYrdoa0C4gtzHF14vsBmdsL7jVmh67at0YJ
y0zBeiS9ioc85_faaQkwth7LoxVAhBJqdsXc6Zj593-
oezG89wBujW6Ty4YT0iOyKiSu1jUrvkCPGNc012Rzd5pjUCna1HNy7cZcrD-
5dT28xBJttELFsz9xGKPY9xW5FaI2r2K9S_FpdNCXoqfzCoVRUED9If8yKKoerKYNrAsaE25gtTjnRJBhKaX4aOXdGKm3y
X6fC32d7nKDEjyiXsmCQ8L4U21iYSqYnxMBZnhNg
```

Теперь необходимо сформировать с этим токеном кубконфиг:

```
# Удаляем стандартного юзера
[root@k8s-istio ~]# kubectl config delete-user user
deleted user user from ivan.kubeconfig
# Удаляем стандартный контекст
[root@k8s-istio ~]# kubectl config delete-context Default
warning: this removed your active context, use "kubectl config use-context" to select a
different one
deleted context Default from ivan.kubeconfig
# Прописываем в конфиге пользователя и его токен
[root@k8s-istio ~]# kubectl config set-credentials ivan --token=$TOKEN
User "ivan" set.
# Создаем контекст и переключаемся на него
[root@k8s-istio ~]# kubectl config set-context Default --cluster=local --user=ivan
Context "Default" created.

[root@k8s-istio ~]# kubectl config use-context Default
Switched to context "Default".

# Проверяем
[root@k8s-istio ~]# kubectl get no
NAME        STATUS   ROLES           AGE   VERSION
k8s-istio   Ready    control-plane   55m   v1.34.6+k0s
```

Все работает!

Если Иван уволится, мы можем в одну команду отозвать его кубконфиг:

```
# Все работает
[root@k8s-istio ~]# kubectl get no
```

```
NAME        STATUS   ROLES           AGE   VERSION
k8s-istio   Ready    control-plane   59m   v1.34.6+k0s

# Удаляем Ивана
[root@k8s-istio ~]# kubectl delete serviceaccount ivan
serviceaccount "ivan" deleted from default namespace

# Токены кешируются - отзыв происходит в течении минуты, но не сразу
[root@k8s-istio ~]# kubectl get no
NAME        STATUS   ROLES           AGE   VERSION
k8s-istio   Ready    control-plane   59m   v1.34.6+k0s

[root@k8s-istio ~]# kubectl get no
NAME        STATUS   ROLES           AGE   VERSION
k8s-istio   Ready    control-plane   59m   v1.34.6+k0s

[root@k8s-istio ~]# kubectl get no
error: You must be logged in to the server (Unauthorized)
```

Для понимания, вот так вот будет выглядеть кубконфиг с авторизацией по токену:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURBRENDQWVpZ0F3SUJBZ0lVTVhZMjhWMEJaQld2Ry9hQnNZdDJ5OV
prRmVRd0RRWUpLb1pJaHZjTkFRRUwKQlFBd0dERVdNQlFHQTFVRUF4TU5hM1ZpWlhKdVpYUmxjeTFqWVRBZUZ3MHlOakEw
TURrd056TTBNREJhRncwegpOakEwTURZd056TTBNREJhTUJneEZqQVVCZ05WQkFNVERXdDFZbVZ5Ym1WMFpYTXRZMkV3Z2
dFaU1BMEdDU3FHClNJYjNEUUVCQVFVQUE0SUJEd0F3Z2dFS0FvSUJBUURYOE9CZW9wN1VyektEbnpjQzRUNFNjZWVKZndG
VFkvZWsKNXZQQjhIWlFWUWhzRzg5dlMrZEdJdkRZK2ZmMjd0YjZCVUs3ajdGd1Jta3ROSFpNSnNuUEVuSEZvdWp6YlkxSQ
o2SUhQVEY2TXE2VVJnU0NGSWtEVWRrd3JTZ0xQSFlKK3Bic0hNSFF2bW81ZFRQV1dCVzlVeVpxSnFzeW80WjBvCjVaS0VH
c01reUFkVFE1b1J4d2Z3MkMwb1YyZ2N2dWxSdWI1aUtCVXBYRTM4ZGkrcEZ6WkNCaEp3NUQ2UU90cXgKUjlydHZtelRmRm
JETlo3WkdEZlM4cFN4RTBwUThQZEF5M2pjQVpacGpueGZvYldueTdYRGVHOFJkQktzMTJMWApFK3RuSE9HRVB3ck5hb1Q1
MTg4R3R3a095dGxvTEgyanlwUTJtT05FMTJxZlMrVzRnR2hsQWdNQkFBR2pRakJBCk1BNEdBMVVkRHdFQi93UUVBd0lCQm
pBUEJnTlZIUk1CQWY4RUJUQURBUUgvTUIwR0ExVWREZ1FXQkJRVGlNbkYKWDhkUWhQYWdzZ2VXRWJNYXpnRDdrVEFOQmdr
cWhraUc5dzBCQVFzRkFBT0NBUUVBYTVjOWl1YUdGNndiUGcwdApBZlRFRXUrbEZXMXlvRU4zWlU4T0lHdjIzMWh3ekRDVV
dmYXlNOTkrcS9hbnlUbU5YMWZ4eExtMGNDMk9lRkJXCnM2bWc2STF1SFJrOTdJRU1FNGZkNFlhNVVoOGR0NTh0Vld3T3JS
S1hoUFFkZWNTVGVISnM2MzBtcktsa1k2aTcKZTZFMHk0UU80cjBRVzN1N1hTMDNZSEpZc0duVCtmYVdSSTBmcEoyWWQ4OT
```

```
A4SS9FbzBmMDR6bjlmYjZOVE5wMApwOHJUeDNpallhek1GNk1seHI1OFJoRmcxa1k5dWZEM1NsdkN5Q1F4Q2hDMUZKelBQ
aVNMcTZFNkt2VFloTHVzCkNTbW54WnRtMkh6Z25BWGE5aVRsN3krNDNQZDM3bEZ2VTJIQnRMczY0YjIrLzNlbjlKL2wwWi
twVUJkVzhxenMKNXdVb0xnPT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://10.13.37.199:6443
  name: local
contexts:
- context:
    cluster: local
    user: ivan
  name: Default
current-context: Default
kind: Config
users:
- name: ivan
  user:
    token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IkdkNmc0RU5lb0ZiT254dWxSQS1yU3B4NGhLMlZIWGVCc3MzYWxMdjRmWHcifQ.eyJ
hdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjIiwic3lzdGVtOmtvbm5lY3Rpdml0eS1zZXJ2ZXIiXSwiZ
XhwIjoxNzc4MzE1NTI0LCJpYXQiOjE3NzU3MjM1MjQsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2YyI
sImp0aSI6IjY2OWM0YmJkLTU4NDUtNDBjYy04ZDEyLTdhYWY0ZTY2Y2U1MiIsImt1YmVybmV0ZXMuaW8iOnsibmFtZXNwY
WNlIjoiZGVmYXVsdCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJpdmFuIiwidWlkIjoiNGVjOWE1YTUtMjE0MS00M2I
zLWE4YjgtYjNmOWE4NzQ3YmJhIn19LCJuYmYiOjE3NzU3MjM1MjQsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZ
WZhdWx0Oml2YW4ifQ.VM9stmAYwvZg-6KDOFd_rD9lqf4RHSh9U-
iahsncKEfUHZzwBYpE3tJ3ZxZvJ3G0STln1BKaioeLKn_MiSIY4ttn8zvlK_wQs-fslOsAAw5mnRYDrD7f-
9H1c4P_RQo8dRULmYiCHIlq3OqqPpJtIUT1qiPEyIotThGxufQ5MEdmKDUExN9HIeaQbZbOX-
lj1kKqETyuhQGrTjVqCs2G7gPSB3vPbgIAB71PxnvChwsK5M3Ijd1VarKN3rq87g-Qpo-
xXLyvNfXyr4_RjUw9_4HokC7HhlOUI_NXYYN0lVzoKJeYVz8CJeIqGAv5CLmlnhvask5rbTGwau7VZStLpw
```