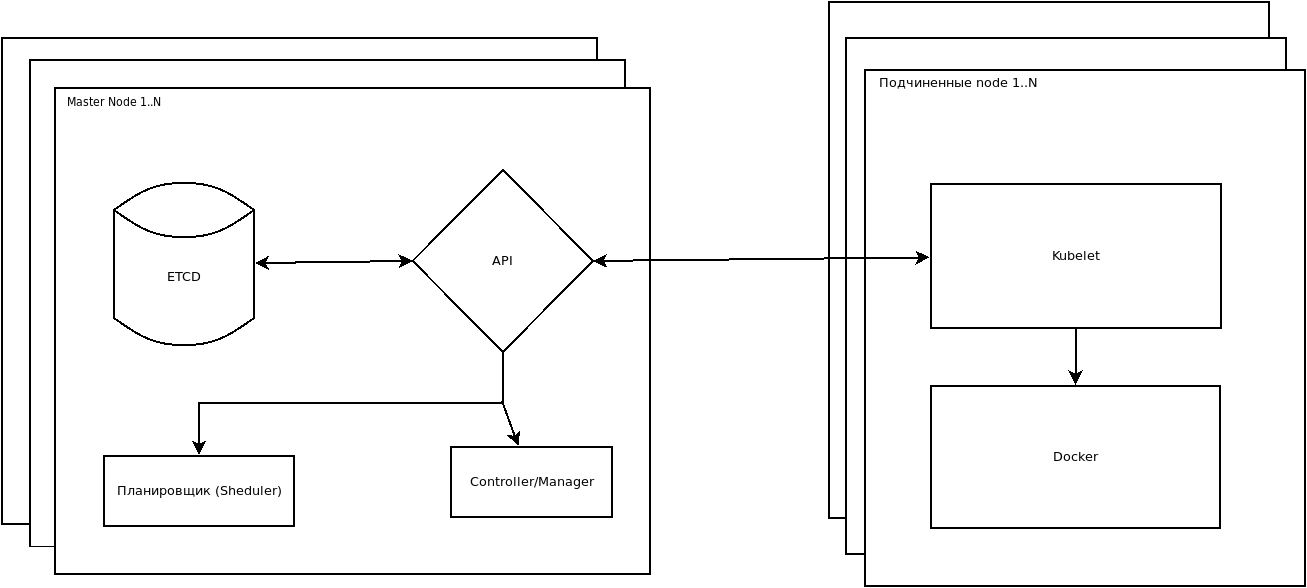
Для полноценного прохождения урока нужно будет 6 компьютеров или виртуальных машин. Будет 3 управляющие ноды и три подчиненные. Желательно чтобы на каждой машине было по 4 ядра и 4Гб оперативки. Более слабые можно. В этом случае возможны подтормаживания.

Думаешь это много? Ну так Kubernetes применяется там, где задачи намного более ресурсоемкие, чем одинокий сайтик или лабораторная работа. И отдать пару крупинок под управление не так и накладно.

Логически Kubernetes можно представить так:



Как видишь ноды делятся на управляющие (master node) и подчинненные.

Как подчиненных нод, так и управляющих несколько. У нас же кластер и управление тоже надо резервировать.

На управляющих нодах тоже можно держать клиентские сервисы, но я не рекомендую. Лучше разделять сущности. Кстати, официальная документация со мной согласна.

А теперь рассмотрим каждый компонент по отдельности и как они действуют совместно.

**API Server**

Как можно увидеть, единым узлом для общения компонентов является API Server.

- он занимается общением с kubectl

- он занимается общением с подчиненными нодами

- он занимается общением с планировщиком и контроллером кластера.

- и он же работает с БД состояний кластера. Изолирует ее от прямого доступа от остальных.

Как уже понятно из названия - для потребителей API Server предоставляет удобный унифицированнный набор команд и методов. И единую точку их использования.

Внутри себя ничего не хранит, так что его можно тиражировать хоть сотнями штук.

И лучше тиражировать, т.к. в единственном числе он является точкой отказа.

Основная сущность, с которой работает API Server это ресурс. Это абстракция, которая включает в себя оперативку, ядра, диски, неймспейсы, сети и т.д.

Различные компоненты кластера могут в рамках определенных им полномочий создавать, удалять или модифицировать имеющиеся ресурсы.

**Etcd**

Это база даннных состояния кластера. Это noSQL БД. Довольно простая. Занимается хранением простых структур вида: ключ - значение. Чем-то напоминает системы кеширования для веб-севера. Но есть и отличия =).

Etcd - это кластерная база данных. В ней согласованность действий является более важной, чем устойчивость к разделению.

По сравнению с аналогами (ZooKeeper, Consul, chubby) etcd очень простая. За это ее и выбрали для проекта kubernetes.

Поскольку во главе угла поставленна согласованность - операции записи всегда идут строго синхронно по порядку. Читать при этом можно из любого экземпляра etcd. Для этого согласований не требуется.

Строгий порядок записи дает возможность производить любые операции с данными, не думая о конфликтах или блокировках.

Но простота это не все, что привлекает в etcd.

Есть в этом ПО механизм наблюдения (watch). Он тоже играет большую роль в работе k8s кластера.

Суть механизма watch в том, что клиент БД может подписаться на определенную ветку с данными и БД известит его об изменениях сразу после их внесения.

Таким образом достигается как скорость обработки изменений, так и экономия трафика в сравнении с обычным регулярным опросом типа polling.

Один компонент кластера записывает что-то в базу, а остальные могут тут же на него среагировать, т.к. им прилетело уведомление с подробностями об изменениях в БД.

Как пример:

- создание pod-а

- запуск или остановка контейнера

- и т.д.

Есть у etcd и недостатки:

Кластер работает по алгоритму raft, а это значит что:

1. Все сообщения за запись шлются лидеру. А уже с него подчиненнным.

2. Для консенсуса нужно чтобы большинство узлов кластера было живо. Иначе кластер может развалиться и потребуется ручное вмешательство.

И некоторые архитектурные недостатки, которые заложены в коде:

1. Это дисковая БД. Все изменения тут же пишутся на диск. Хранение в оперативке минимально. Соответственно нужны быстрые диски. Лучше SSD.

2. Есть ограничение на размер ключа в паре ключ-значение. Максимальный размер 1Мб.

Хотя может это и благо, т.к. огромные куски данных будут замедлять работу.

3. Есть ограничение на максимальный размер всей БД. По умолчанию 2Гб, но через настройки можно увеличить до 8Гб. Для кубернетиса сойдет, а так вообще мало в 21-м веке ).

**Controller manager и Sheduler**

Они отпочковались от API Server в отдельные сервисы, но все еще очень тесно взаимодействуют. Поэтому я их описываю вместе.

Sheduler занимается регламентными работами внутри кластера:

- проверяет достоверность кэшированных даннных о состоянии кластера

- анализирует поды. В том числе находит "непривязанные" поды и отдает запрос на поиск для их места в кластере.

- привязывает и перепривязывает под к ноде

и т.д.

Эдакий секретарь-делопроизводитель.

Контроллер следит за работой уже привязанных pod-ов, реагирует на запросы от планировщика по части модификации подов, создает/удаляет под-ы, запрашивает и выдает им ресурсы и т.д.

Что-то вроде исполнительного директора =)

**Kubelet**

И еще один компонент, который нужно рассмотреть это kubelet. Kubelet выполняет роль "агента" kubernetes. Через него осуществляется взаимодействие с системой контейнеризации (сейчас это докер, но возможно когда-то будет поддерживать и другие технологии).

Он получает данные о том, что у него на ноде должно быть запущено и запускает. Также отдает информацию о состоянии сервисов на ноде для контроллера.

Вот такая вот система. Вроде и сложная, а вроде и простая. Но в любом случае, если на тебя вывалили большой объем информации, тебе может показаться что ничего непонятно.

Поэтому я распишу по шагам одну из "рабочих ситуаций" кластера - создание нового пода.

1. Пользователь с помощью API Server создает под, и API Server записывает изменения в etcd.
2. Планировщик замечает непривязанный (unbound) под и определяет, на какой ноде его запустить. Эта связь записывается в API Server.
3. Kubelet замечает произошедшее в наборе привязанных к его ноде подов изменение и запускает контейнер в Docker.
4. Kubelet отслеживает статус пода с помощью среды выполнения. Если что-то меняется, он уведомляет об этом API Server.

На этом теоретический блок практически закончился.

Все описанные компоненты можно запускать как на живом железе, так и в контейнерах.

Однако как минимум etcd я рекомендую запускать на голом железе. Так будет надежнее и быстрее (трафик от контейнеров идет через linux-bridge, который грузит процессор и может давать лишние задержки).

А теперь мы с тобой на практике поднимем кластер kubernetes на 6-ти нодах (ты ведь уже подобрал подходящие физические или виртуальные машины? если нет, то самое время).

В качестве ОС я буду использовать Debian Linux. Думаю для убунты порядок действий будет таким же. Так что я не ограничиваю тебя выбором одного дистрибутива.

Адреса и имена машин будут такими:

master-node-1 10.0.0.1

master-node-2 10.0.0.2

master-node-3 10.0.0.3

slave-node-1 10.0.0.4

slave-node-2 10.0.0.5

slave-node-3 10.0.0.6

Начнем с установки etcd, как ключевого элемента. Дальше вокруг нее поднимется и остальное окружение. Мы с тобой уже опытные системные администраторы и поэтому я не буду полностью покомандно расписывать все действия. Но я дам достаточно информации, чтобы ты смог все настроить.

Я не буду против (а буду даже за), если ты воспользуешься средствами автоматизации ;). Ведь придется выполнять однотипные операции на группе хостов. Лично я воспользуюсь ansible и в конце курса поделюсь полученным плейбуком.

Дальше

Итак. Etcd. Для установки всего окружения требуется не так много действий.

**sudo apt-get install etcd**

В интернете есть много мануалов, где для установки etcd запускают неведомые скрипты из интернета. Ты можешь сделать и так, но обязательно сначала прочти этот скрипт и убедись, что он не делает ничего лишнего ;).

Думаю уж с установкой то ты справился на отлично. Теперь нужно составить конфигурацию etcd всех трех узлов master. Если ты, как и я ставил пакет из официального репозитория, то конфигурационный файл будет располагаться в **/etc/default/etcd**.

Нужно будет заменить его следующим содержимым:

***ETCD\_NAME="{{ ansible\_hostname }}"***

***ETCD\_LISTEN\_CLIENT\_URLS="http://0.0.0.0:2379"***

***ETCD\_ADVERTISE\_CLIENT\_URLS="http://{{ ansible\_ssh\_host }}:2379"***

***ETCD\_LISTEN\_PEER\_URLS="http://0.0.0.0:2380"***

***ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://{{ ansible\_ssh\_host }}:2380"***

***ETCD\_INITIAL\_CLUSTER\_TOKEN="EtcdKuberCluster"***

***ETCD\_INITIAL\_CLUSTER="master1=http://10.0.0.1:2380,master2=http://10.0.0.2:2380,master3=http://10.0.0.3:2380"***

***ETCD\_INITIAL\_CLUSTER\_STATE="new"***

***ETCD\_DATA\_DIR="/var/lib/etcd"***

***ETCD\_ELECTION\_TIMEOUT="5000"***

***ETCD\_HEARTBEAT\_INTERVAL="1000"***

Это шаблон ansible. Ты можешь заметить там несколько переменых. Их надо будет заменить для каждого из хостов.

**{{ ansible\_hostname }}** - это хостнейм узла

**{{ ansible\_ssh\_host }}** - это ip-адрес узла

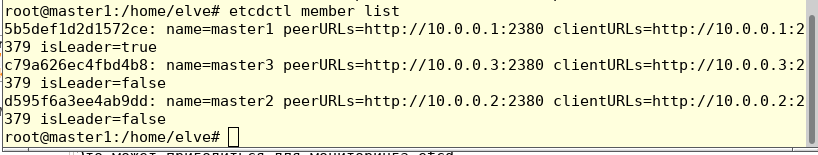
Тут мы задаем адреса, на которых демон будет принимать входящие соединения клиентов и соседей по кластеру.

После модификации конфигов, перезапускаем сервис

**sudo systemctl restart etcd**

Теперь можно проверить, что кластер работает. Т.к. база пустая, то сойдется он практически мгновенно. Проверка выполняется с любой из нод вот таким образом:

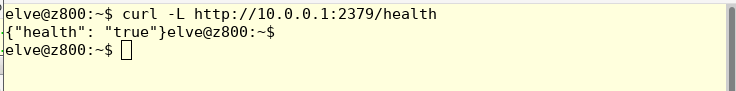
**etcdctl member list**



Это значит, что кластер работает. Видны все 3 ноды. А также, если присмотришься, то увидишь кто из них лидер, а кто подчиненные =).

Проверку состояния каждой из нод можно проверить так:

**curl -L http://10.0.0.1:2379/health**



Это может пригодиться для мониторинга etcd.

Утилита **etcdctl** может очень многое. С ее помощью можно диагностировать и обслуживать кластер и даже работать с данными, если очень надо это делать именно из консоли.

Есть правда одна сложность. Есть несколько наборов команд (API) для работы через **etcdctl.** На данный момент актуальна 3-я версия API. Чтобы использовать ее надо в консоли в сетевом окружении объявить переменнную **ETCDCTL\_API=3**

**export ETCDCTL\_API=3**

Сейчас база пуста. Можно немножко поиграться.

Давай создадим пару ключ-значение.

**etcdctl put test on**

**etcdctl put test2 on**

**etcdctl put test4 gogogo**

А теперь прочтем эти значения.

Вот так можно посмотреть конкретную переменнную:

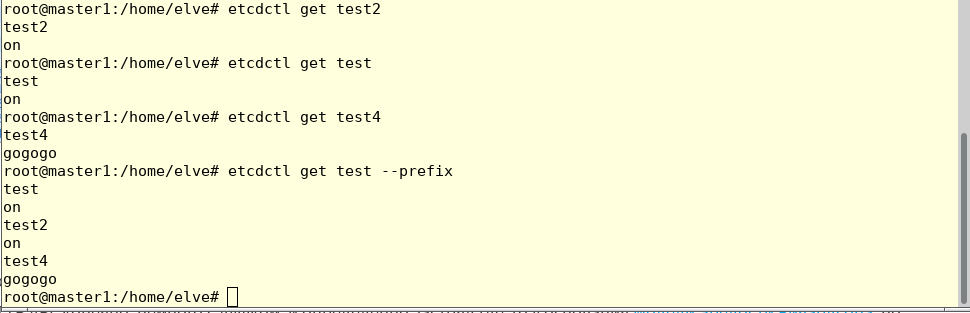
**etcdctl get test2**

или

**etcdctl get test4**

или

**etcdctl get test --prefix**



Эти операции можно делать с любой из нод. Все равно сработает.

В последнем варианте я запросил все переменнные, чьи названия начинаются с test

Для скриптов будет очень удобно, если выводиться будет только значение переменной. Для этого есть свой флаг:

**etcdctl get test4 --print-value-only**

Ну раз можно переменные просматривать, то можно и удалять.

**etcdctl del test**

Ну и конечно очень важная для отладочных целей функция

**etcdctl watch <параметры> <имя\_переменнной>**

Это слежение за изменениями переменных в реальном времени.

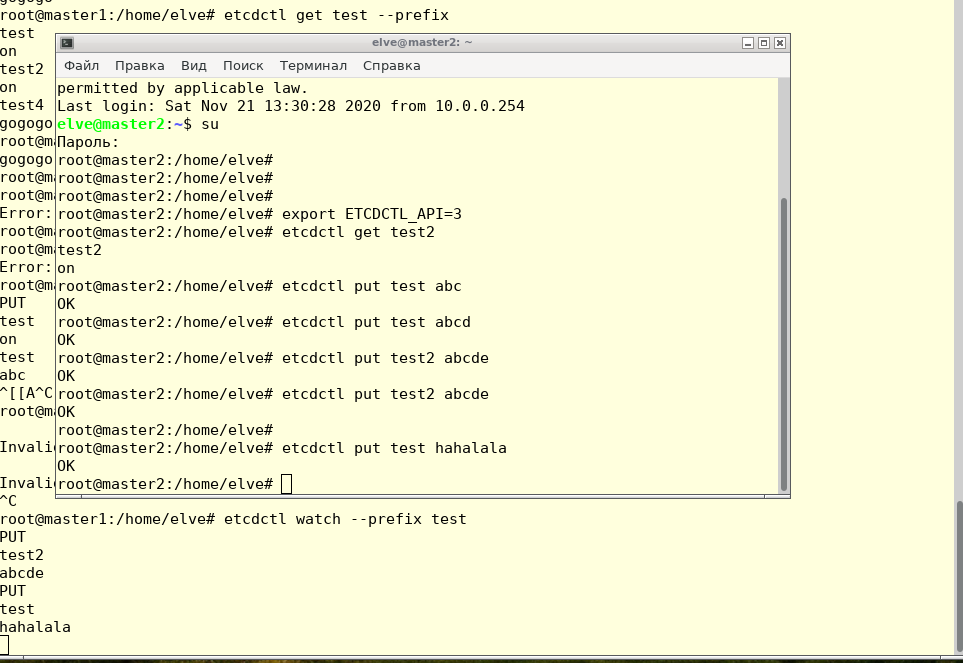
К примеру вот наблюдение за переменной test2

**etcdctl watch --prev-kv test2**

А вот так можно следить по префиксу

**etcdctl watch --prefix test**

Для эксперимента на одной из нод ты можешь запустить watch, а с другой попробовать подобавлять и поизменять переменные. Вот как на картинке.



Если ты не разработчик, то этого набора на первое время тебе вполне хватит.

Но это все игрушки. А теперь немного об обслуживании нашей БД.

**Дефрагментация**

Первое что надо знать. Базе etcd требуется периодическая дефрагментация. Если активно пользоваться базой, то в ней возникают проблемы схожие с проблемой файловой системы. И начинаются подтормаживания. Поэтому я бы раз в сутки выполнял дефрагментацию всего кластера. Для этого нужно по крону запускать на каждой ноде следующую команду:

**etcdctl defrag**

Если база разрослась, а ты давно не делал дефрагментацию, то процесс может занять продолжительное время. Также, если от дефрагментации начнутся сильные задержки, то кластер может перевыбрать лидера ). Обычно это не страшно, но изредка бывает что-то вроде "шторма перевыбора лидера". Это когда ни одна из нод не отвечает вовремя и они не могут между собой определиться кто главный =). Тут возможно придется вмешиваться руками.

**Сжатие**

В данном случае сжатие не подразумевает сжатие данных с помощью специальных алгоритмов. Тут речь немного о другом. Etcd хранит историю изменения значений переменных. И сжатие означает удаление старых записей из истории. Тем самым освобождается дисковое пространство. Ты же помнишь, что максимальный размер базы 8Гб?

Чтобы узнать номер нынешней ревизии, надо выполнить вот такую команду:

**etcdctl --endpoints=:2379 endpoint status --write-out="json" | egrep -o '"revision":[0-9]\*' | egrep -o '[0-9]\*'**

оно выдаст цифру. Эту цифру можно использовать для чистки:

**etcdctl compact <цифра>**

Время от времени надо это делать. Конкретный режим выбирай от нагрузки.

А вот представим, что ты проспал все события и у тебя **etcd** начал странно себя вести. Как узнать что его тревожит?

Вот эта команда покажет тебе общую информацию о конкретной ноде:

**etcdctl --write-out=table endpoint status**

А вот этой можно увидеть на что жалуется наш пациент:

**etcdctl alarm list**

Когда ты исправишь проблему, аларм сам собой не уйдет и база в штатный режим не перейдет. Чтобы вернуть базу в строй надо будет ввести вот эту команду:

**etcdctl alarm disarm**

На этом я думаю закончить знакомство с Etcd и двигаться дальше. Если тебе интересно поглубже разобраться, то можешь это сделать в официальной документации: <https://etcd.io/docs/v3.2.17/op-guide/>

И вот когда кластер Etcd у тебя настроен пора переходить к установке остальных компонентов Kubernetes.

Начинается легкий экшн ;)

Сейчас мы установим компоненты kubernetes и настроим их.

Предварительно придется произвести кое-какую подготовку операционной системы на нодах.

Я очень надеюсь, что ты взрослый мальчик и проходишь этот курс не с нулевыми знаниями. Поэтому не все побочные действия буду расписывать покомандно.

Ты должен и так знать как работать с файрволом и как копировать файлы между серверами.

**1. br Tables**

Включим **br\_iptables**. Кусок файрвола, позволяющий ему управлять L2 трафиком внутри бриджей. Это нужно для управления сетью **kubernetes**.

Делается это так:

**cat <**

**net.bridge.bridge-nf-call-ip6tables = 1**

**net.bridge.bridge-nf-call-iptables = 1**

**EOF**

**sudo sysctl –system**

**2. Swap**

После этого отключим на машине своп. При запуске сервисов, все равно будет ругать и придется выключать. Так что лучше сразу озаботиться =). На всех нодах выполняешь:

**sudo swapoff -a**

и потом в /etc/fstab нужно закомментировать строчку про своп, чтобы после перезагрузки он не всплыл случайно.

Если на ноде будет включен своп - кластер откажется собираться.

**3. Firewall**

Теперь, пока мы не закопались в настройках - проверь файрвол. Для мастер-ноды должны быть открыты следующие порты:

***6443/tcp Kubernetes API***

***2379-2380/tcp Etcd***

***10250/tcp Kubelet API***

***10251/tcp KubeScheduler***

***10252/tcp KubeController***

Для подчиненной ноды надо открывать следующие порты:

**10250/tcp KubeletAPI**

**30000-32767 NodeServices**

Все проверил и настроил? Хорошо. Переходим к установке. По сути надо подключить репозиторий и поставить оттуда софт. Вот рекомендуемая последовательность действий в официальной документации (https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#)

**sudo apt-get update && sudo apt-get install -y apt-transport-https curl**

**curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -**

**cat <<="" strong="">**

**deb https://apt.kubernetes.io/ kubernetes-xenial main**

**EOF**

**sudo apt-get update**

**sudo apt-get install -y kubelet kubeadm kubectl**

**sudo apt-mark hold kubelet kubeadm kubectl**

К сожалению репозиторий поддерживается абы как. Именно поэтому для всех deb-дистрибутивов репозиторий kubernetes-xenial (хотя можно было хоть симлинками привести к нормальному виду). Главное что все что установится будет работать =).

И докер поставим самый свежий. Можно конечно из репозитория мне в плейбук пару лишних строк добавить было не жалко =)

**curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -**

**apt-key fingerprint 0EBFCD88**

**add-apt-repository \**

**"deb [arch=amd64] https://download.docker.com/linux/debian \**

**$(lsb\_release -cs) \**

**stable"**

**apt-get update**

**apt-get install docker-ce docker-ce-cli containerd.io**

У докера репозиторий сделан правильно и поэтому указываем имя дистрибутива и имя релиза.

Вот теперь все установлено и прописано и пришло время поднять кластер. Теперь будем действовать руками. Ansible нам больше не помощник. Механизм такой:

- сначала инициализируем и поднимаем первую ноду. при этом генерируются ssl-сертификаты, нужные для общения нод между собой

- далее копируем сертификаты и по очереди подключаем каждую ноду к первой.

Выглядит несложно. Давай попробуем =).

Для инициализации ноды нам потребуется первоначальная конфигурация. В документации конечно много всего интересного написано, но я все же приведу пример своего конфига. Как мне кажется он минимально возможный.

***apiVersion: kubeadm.k8s.io/v1beta1***

***kind: InitConfiguration***

***localAPIEndpoint:***

***advertiseAddress: 10.0.0.1***

***---***

***apiVersion: kubeadm.k8s.io/v1beta1***

***kind: ClusterConfiguration***

***kubernetesVersion: stable***

***apiServer:***

***certSANs:***

***- 127.0.0.1***

***- 10.0.0.1***

***- 10.0.0.2***

***- 10.0.0.3***

***controlPlaneEndpoint: 10.0.0.1***

***etcd:***

***external:***

***endpoints:***

***- http://10.0.0.1:2379***

***- http://10.0.0.2:2379***

***- http://10.0.0.3:2379***

***networking:***

***podSubnet: 10.11.0.0/16***

Сохраняй его в файл под именем **kubeadm-init.yaml**. Либо сразу на мастер-ноде-1, либо просто скопируй его туда.

Да. Это YAML. В случае с докером и все что с ним связанно этот формат будет нас преследовать.

Давай разберем конфиг повнимательнее. Там в принципе и так примерно по смыслу понятно, но все же.

**apiVersion: kubeadm.k8s.io/v1beta1** - версия api в настройках API Server

**kind: InitConfiguration** - указывает что это конфиг для инициализации, а не рабочий.

**localAPIEndpoint:** - тут указывается адрес на котором будет слушать API Server (мы же помним что у сервера этих адресов обычнно несколько).

Этот кусок был нужен, чтобы поднять API Server и начать конфигурирование. Дальше пойдет уже рабочая часть, которая осядет в настройках.

Идем дальше.

**apiServer: certSANs:** - тут указываются адреса всех нод. Для них будут сгенерированы ssl-сертификаты.

**controlPlaneEndpoint:** - адрес мастера или балансировщика нагрузки. Логично, что указываем себя же ).

**etcd:** - параметры подключения к etcd

**networking:** - настройки сети для pod-ов.

Конфиг рассмотрели. Двигаемся дальше. Не забудь поправить его под себя, если потребуется.

Так. Теперь на мастре-ноде-1 надо выполнить команду (ты ведь скопировал конфиг на мастер-ноду? ;) ):

**kubeadm init --config=$HOME/kubeadm-init.yaml**

Сервер немного задумается. Не переживай. Он не завис. Сейчас он заполняет базу etcd (и изменения разлетаются по нодам), генерирует ssl-сертификаты, скачивает docker-image своих компонентов. В конце-концов все закончится и выдаст примерно вот такое:



От фразы Your kubernetes весь текст сохрани в текстовый файл. Он нам еще пригодится.

Все. У тебя есть первая живая мастер-нода. Все компоненты запущены в докере. Можешь проверить:

**docker ps**



Смотри на столбец Names. Там описано что это за контейнеры.

Раз что-то уже запущено, значит надо настроить и управление.

В тексте, который мы сохранили предлагалось выполнить вот такой набор команд:

**mkdir -p $HOME/.kube**

**sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

**sudo chown $(id -u):$(id -g) $HOME/.kube/config**

Выполни его и для root и для пользователя под которым обычно заходишь. Это настройка kubeadm.

Теперь пришло время поднять остальные мастер-ноды.

Для начала скопируем сертификаты. Для этого нужно просто скопировать папку **/etc/kubenetes/pki** на все остальные ноды.

**scp -r /etc/kubernetes/pki master2:/etc/kubernetes**

**scp -r /etc/kubernetes/pki master3:/etc/kubernetes**

Папка такая в etc уже должнна была создаться. Если нет, то ты где-то накосячил на предыдущих шагах.

Думаю скопировалось все удачно. И раз сертификаты раскиданы, то можно вводить ноды в кластер. По очереди.

Теперь копируй **kubeadm-init.yaml** с первой ноды на остальные две. На каждой ноде в этом конфиге надо будет поменять два параметра:

- localAPIEndpoint: advertiseAddress

- controlPlaneEndpoint

Туда вписывается ip-адрес ноды.

После этого стандартно выполняем команду:

**kubeadm init --config=$HOME/kubeadm-init.yaml**

И настройка kubeadm. Не забывай про нее.

**mkdir -p $HOME/.kube**

**sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

**sudo chown $(id -u):$(id -g) $HOME/.kube/config**

И вот подошел волнительный момент. Есть три ноды в кластере. Наверное.... Как можно в этом убедиться наверняка? Вводи команнду:

**kubectl get nodes**



Вот тут видно, что ноды добавлены. Но кластер пока неполноценный, так что статус NotReady.

Так. Ты можешь управлять кластером с любой из мастер-нод. А что если ты хочешь с рабочего ноутбука управлять? Это тоже возможно.

Нужно установить на него kubeadm также, как мы с тобой выше это делали на нодах.

А после этого с любой из нод копируем файл **/etc/kubernetes/admin.conf** к себе на ноутбук в папку **$HOME/.kube/config.** Очевидно папку **.kube** перед этим придется создать =). Хотя если ты поднимал на ноутбуке minicube, то не придется. Просто перезапиши файл config

Проверить можно той же командой

**kubectl get nodes**

С мастер-нодами закончили. Пришло время подчиненных.

На слейв-нодах нужно также как на мастер-нодах установить docker и утилиты kubectl, kubeadm и т.д. В принципе весь набор действий, кроме установки etcd.

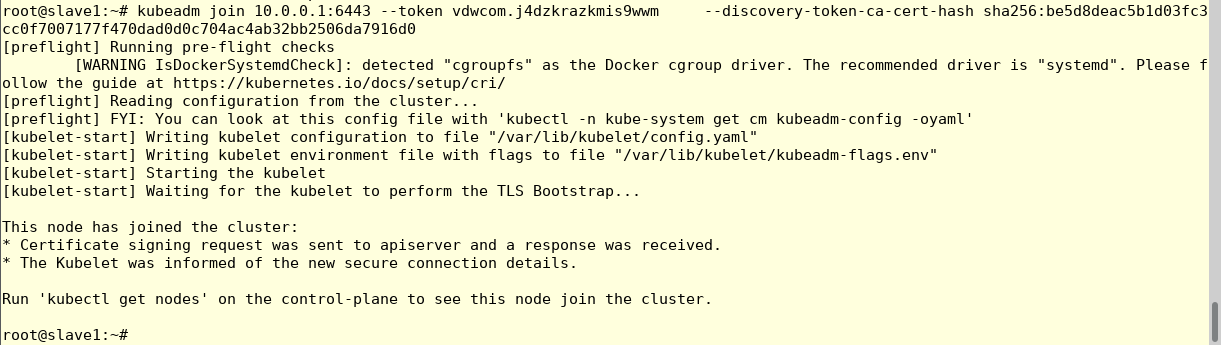
Как это делать ты помнишь.

Когда весь софт установлен можно сразу же включать ноду в кластер. Делается это командой из сообщения после активации мастер-ноды. Если ты уже удалил эти данные, то ничего страшного. Запросить новый токен вместе с подсказкой можно на любой мастер-ноде (или на рабочем ноутбуке) вот такой командой:

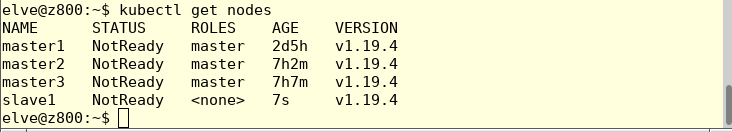
**kubeadm token create --print-join-command --ttl=0**

К примеру у меня получилась такая команда:

**kubeadm join 10.0.0.1:6443 --token vdwcom.j4dzkrazkmis9wwm --discovery-token-ca-cert-hash sha256:be5d8deac5b1d03fc3cc0f7007177f470dad0d0c704ac4ab32bb2506da7916d0**



Все. Можно проверять список нод. Должна появиться еще одна.



И вот у нас уже ситуация в кластере как в гос. организациях большинства стран =). 3 начальника и один подчиненный =)))).

Остальные подчиненные ноды накидывай самостоятельно. Механизм ты понял.

Сервисы docker и kubelet на всех нодах должны быть добавлены в автозагрузку. Иначе после перезагрузки машина не вернется в кластер.

На мастер-нодах еще и сервис etcd надо в автозагрузку.

Кластер у нас с тобой получился. Но не совсем полноценный. Все ноды Not Ready, хоть и онлайн.

Сейчас я покажу тебе как добить его до рабочего состояния. А объяснять буду в другой раз.

Чтобы кластер ожил надо научить его работать с сетью. Это делается через специальные плагины.

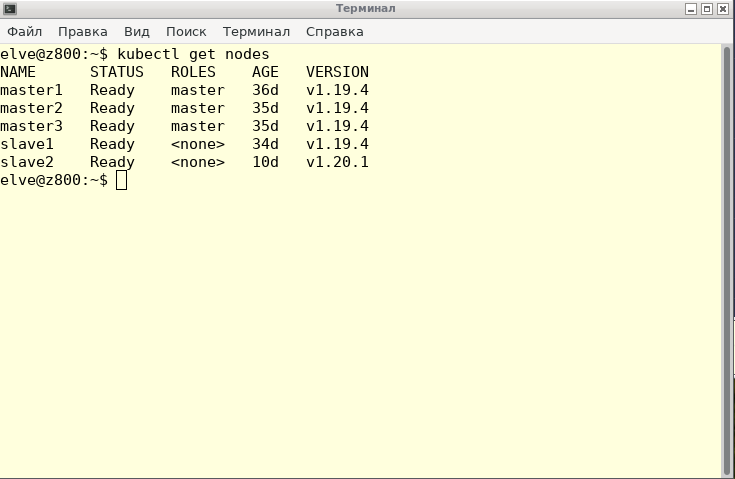
Я за тебя выбрал плагин Calico. Возможно за свою практику другие тебе и не потребуются.

Итак. Установка плагина (выполнять на мастер-ноде):

**kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml**

Ноды немного пошуршат жесткими дисками, т.к. под этот плагин будут качаться image дополнительные с интернета.

И минуты через 2-3 твой кластер станет Ready.



Все. Кластер в боевом состоянии. Можешь уже пользоваться, либо продолжить обучение дальше.