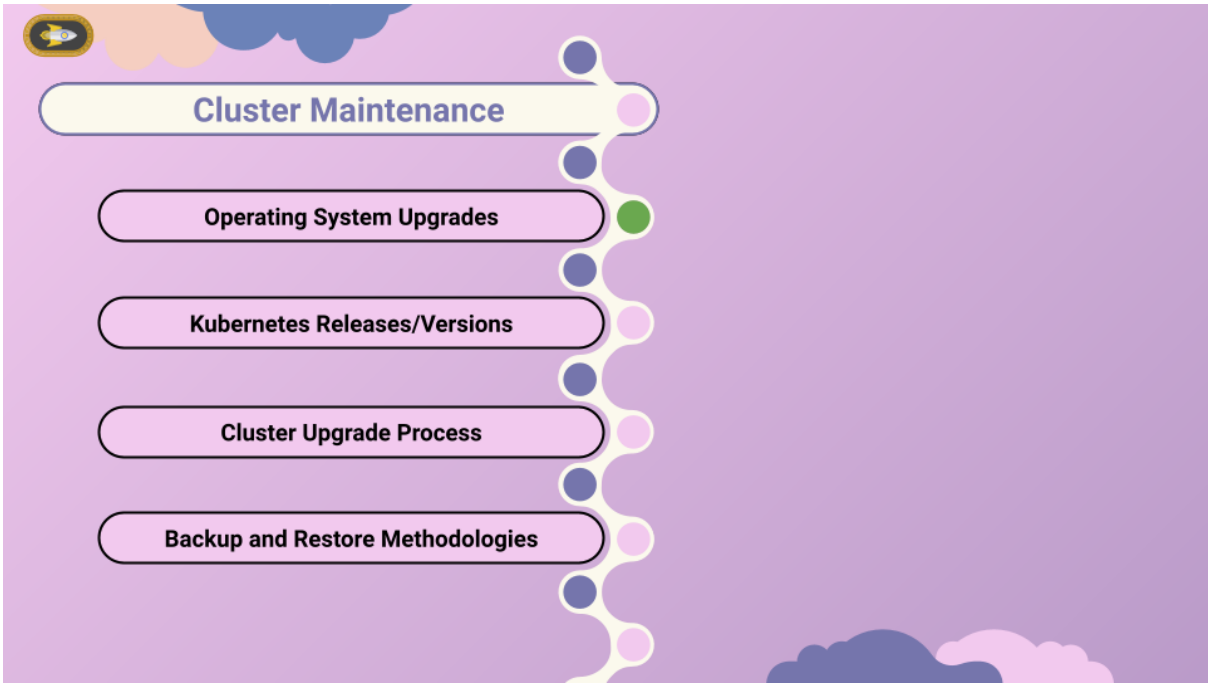


Привет и добро пожаловать на лекцию. В этой лекции мы обсудим сценарии, в которых нам, придется отключить узел от кластера, например, для целей обслуживания.

Это могут быть задачи вроде обновления базового программного обеспечения или патчинг, такой как исправления безопасности и т. д..

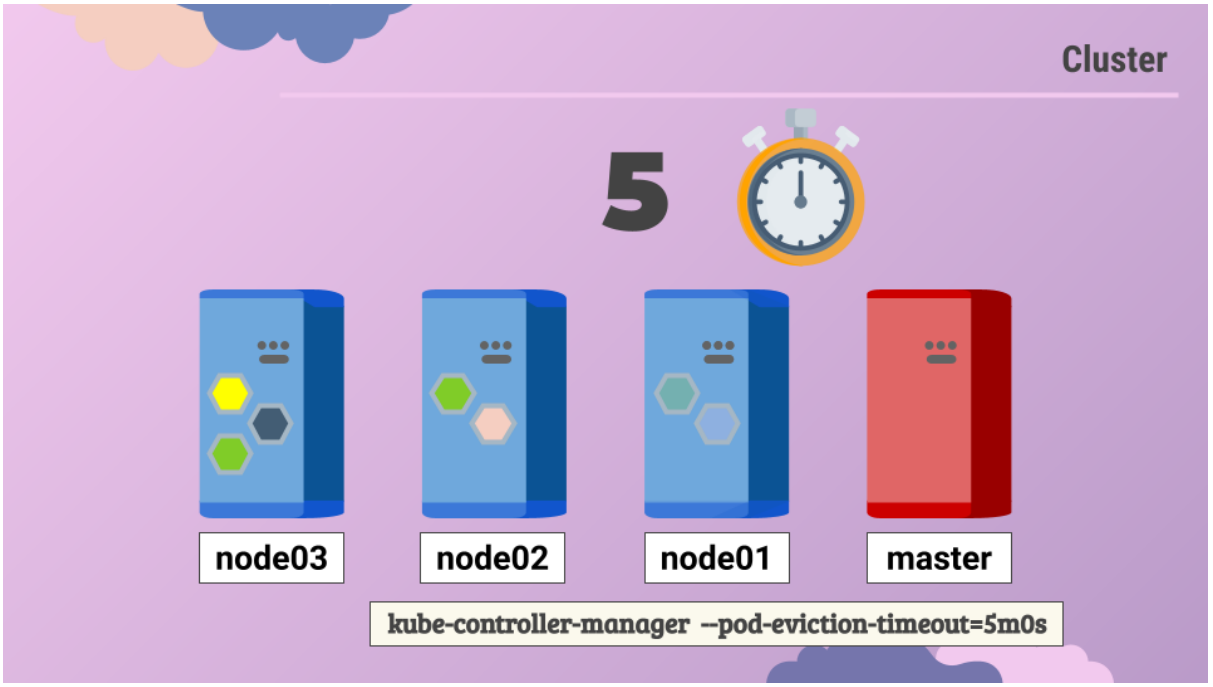
В этой лекции мы увидим разные варианты, доступные для обработки таких случаев.

Итак, у нас есть кластер с несколькими нодами и PODs, с работающими приложениями.

Что происходит, когда один из этих узлов выходит из строя?

Конечно, PODs на них становятся недоступны. Теперь, в зависимости от того, как развернулись эти PODs, это может повлиять на наших пользователей.

Например, поскольку у нас есть несколько реплик зеленых PODs, пользователи, обращающиеся к зеленому приложению, не пострадают, поскольку они обслуживаются через другой зеленый POD, который остался онлайн в данный момент.
Однако пользователи розового приложения пострадают, поскольку единственный розовый POD был запущен на пострадавшей ноде.



Что же в этом случае делает Kubernetes?
Если узел вернулся и стал онлайн сразу, то процесс kubelet запускается и приводит все в надлежащий вид, а PODs снова включаются.

Однако, если узел не работал более 5 минут, PODs прекращают работу на этом узле, потому что Kubernetes считает их мертвыми. Если этот умерший POD являются частью replicaset, он будет воссоздан на других узлах.

Время ожидания возврата POD в оперативный режим называется `pod-eviction-timeout` и устанавливается в controller-manager с дефолтным значением в 5 минут.
Таким образом, всякий раз, когда нода отключается, мастер-нода ожидает до 5 минут, прежде чем считать этот рабочий узел умершим. Когда узел возвращается в строй по истечении pod-eviction-timeout, он становится пустым, и на него не запланированы никакие PODs.

Таким образом, если перед тобой стоят задачи обслуживания, которые необходимо выполнить на узле, и тебе известно, что рабочие нагрузки, запущенные на нем, имеют другие реплики, то нет ничего страшного если, они отключатся от кластера на короткий период времени. Т.е. если ты уверен, что нода вернется в строй в течение пяти минут, то можешь быстро обновить ее и перезагрузить.

Но обычно мы точно не знаем, сколько времени займет техническое обслуживание и уложимся или мы в пять минут. Как правило, мы даже не можем точно сказать, вернется ли вообще узел после наших манипуляций.

Для таких целей есть более безопасный способ. Мы можем целенаправленно удалить с ноды все рабочие нагрузки, так, чтобы они были перенесены на другие узлы в кластере.

Мы используем команду kubectl drain, а дальше имя ноды.

Фактически, это штатные события в кластере, когда PODs аккуратно завершаются на одной ноде и воссоздаются на другой. Узел также будет заблокирован или помечен как не подлежащий планированию. Это означает, что на этот узле нельзя будет назначить какие-либо PODs, пока мы специально не снимем это ограничение.

Теперь, когда PODs переселены и находятся в безопасности на других узлах, мы можем перезагрузить первый узел. Когда он снова появится в сети, на него все еще нельзя будет назначать нагрузку.

Поэтому нам нужно вернуть узел обратно, для этого используй команду

kubectl uncordon node01

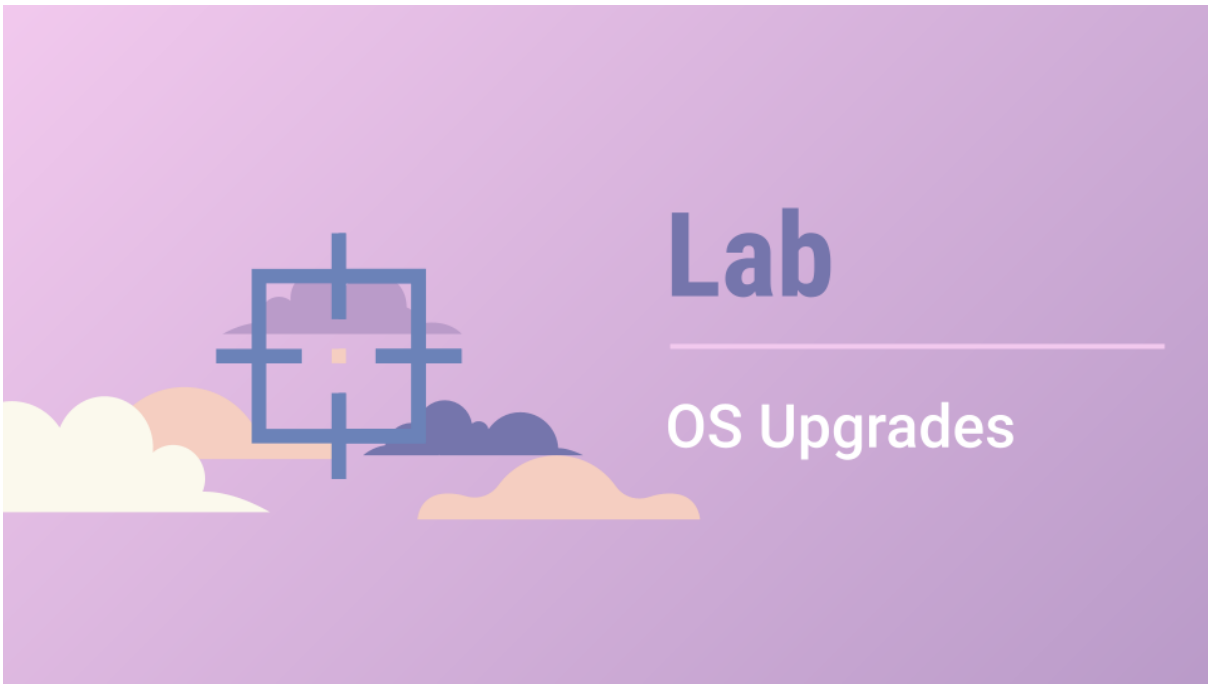
после которой scheduler снова начнет назначать туда PODs.

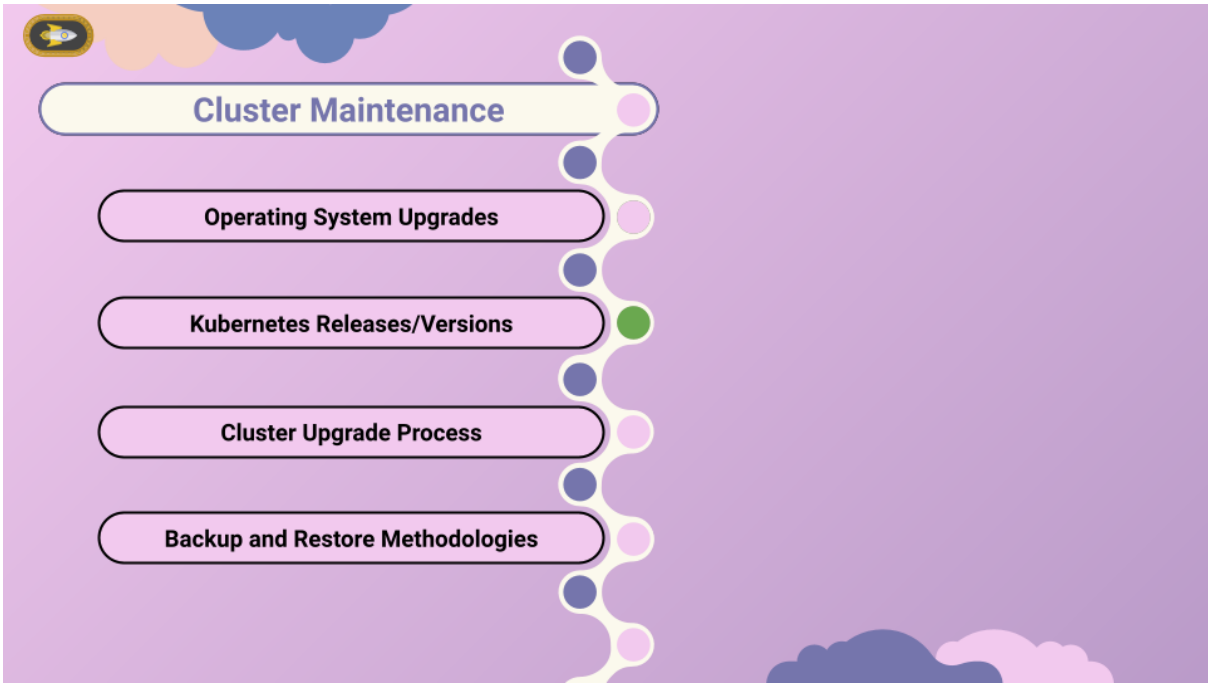
Имей в виду, что PODs, которые были перенесены на другие ноды так и остаются исполняться на нодах, куда они были выселены и не возвращаются автоматически.

Только если какие-то из этих PODs будут удалены и заново созданы, планировщик поставит их на этот освободившийся узел, и только тогда они смогут на нем работать.

Помимо drain и uncordon, существует еще одна команда - cordon. Эта команда просто отмечает узел как не подлежащий планированию, и в отличие от drain, она не завершает рабочие нагрузки и не переселяет PODs с целевого узла.
Она просто запрещает планирование PODs на ноде.

Ну вот и все в этой лекции. Отправляйся на практику и потренируй отцепление нод от кластера. Увидимся на следующей лекции.

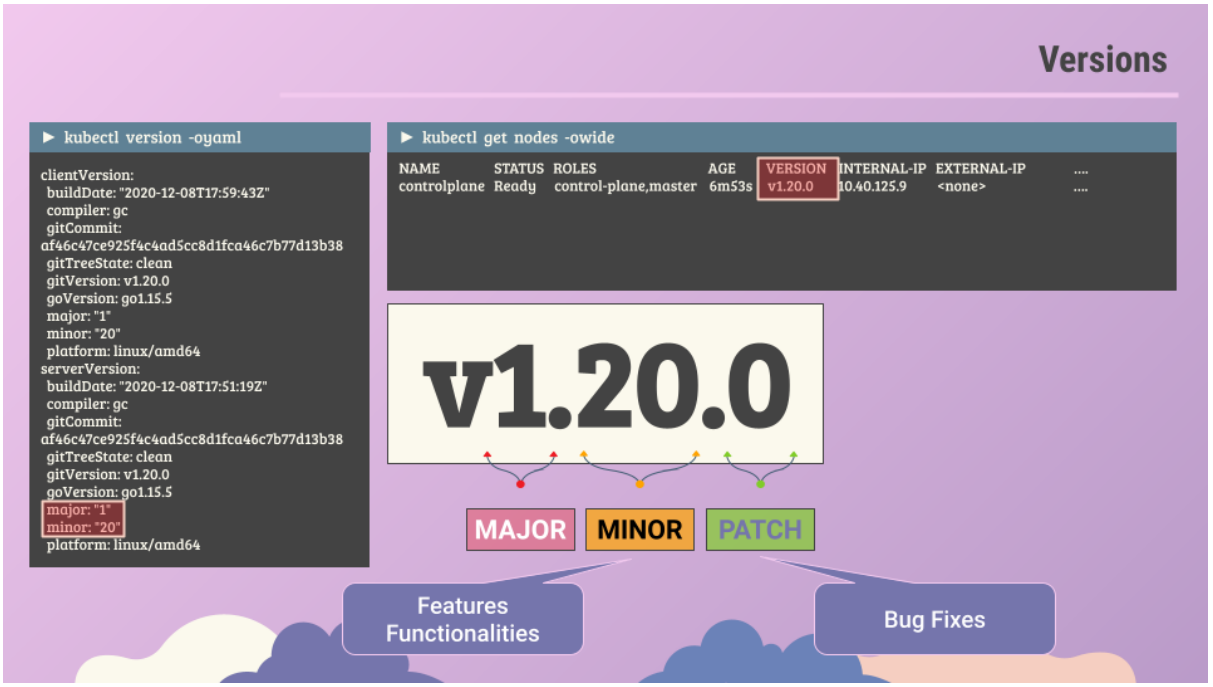




Привет и добро пожаловать на лекцию. В этой лекции мы обсудим различные релизы и версии Kubernetes.

Итак, что мы знаем о версиях в Kubernetes на данный момент? Мы знаем, что при установке кластера мы устанавливаем определенную версию Kubernetes. Мы видим это, когда запускаем команду kubectl get nodes.

В данном случае это v1.20.0. В этой лекции мы увидим, как проект Kubernetes управляет выпусками программного обеспечения. Давай подробнее рассмотрим этот номер версии.



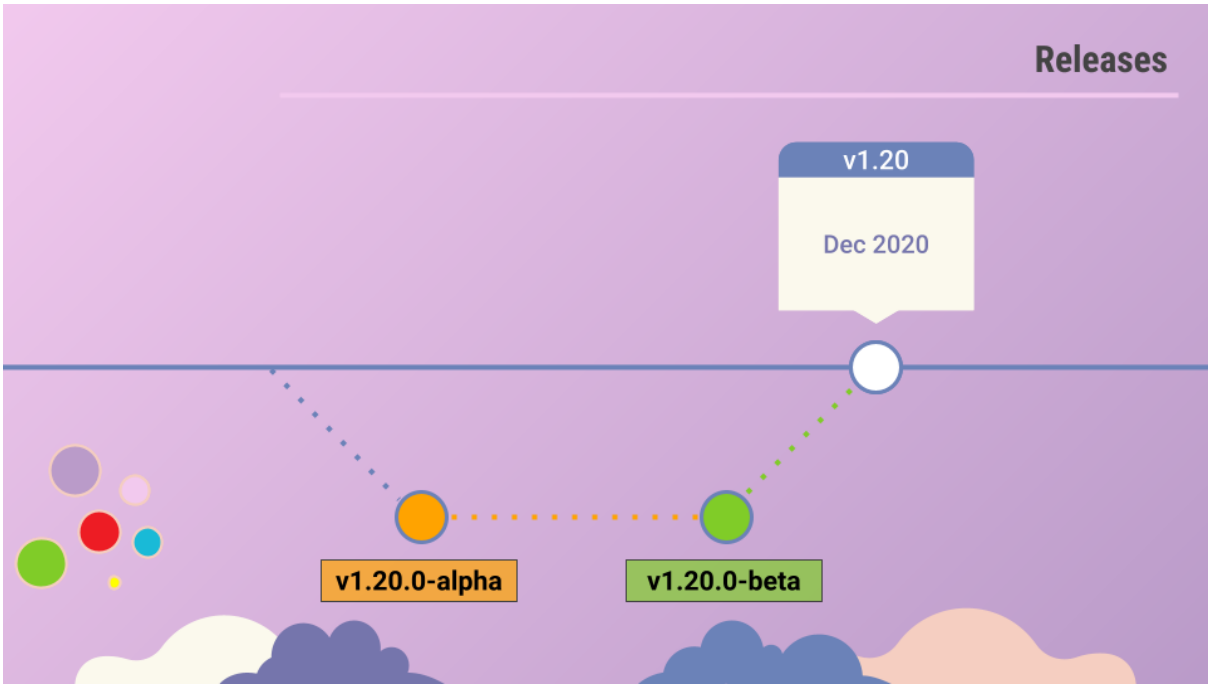
Релизные версии Kubernetes состоят из 3 частей. Первая - это основная - мажорная версия, за ней следует дополнительная - минорная версия, а затем - версия исправлений. В то время как минорные версии с новыми функциями и фичами выпускаются каждые несколько месяцев, патчи, содержащие исправления критических ошибок, выпускаются чаще. Как и многие другие популярные приложения, Kubernetes следует стандартной процедуре управления версиями программного обеспечения. Каждые несколько месяцев он выходит с новыми функциями и фичами через минорный релиз.

Первая мажорная версия 1.0 была выпущена в июле 2015 года.

На момент записи последней стабильной версией является 1.20.0. Все, что мы здесь видели, это стабильные выпуски Kubernetes.

Помимо этого, мы также видим альфа- и бета-версии. В них все исправления ошибок и улучшения сначала переходят в альфа-версию с меткой «альфа» в этом релизе.

В альфе все новые фичи по умолчанию отключены, т.к. могут содержать ошибки. Затем они попадают в бета-версию, там код уже хорошо протестирован, и там новые функции уже включены по умолчанию.



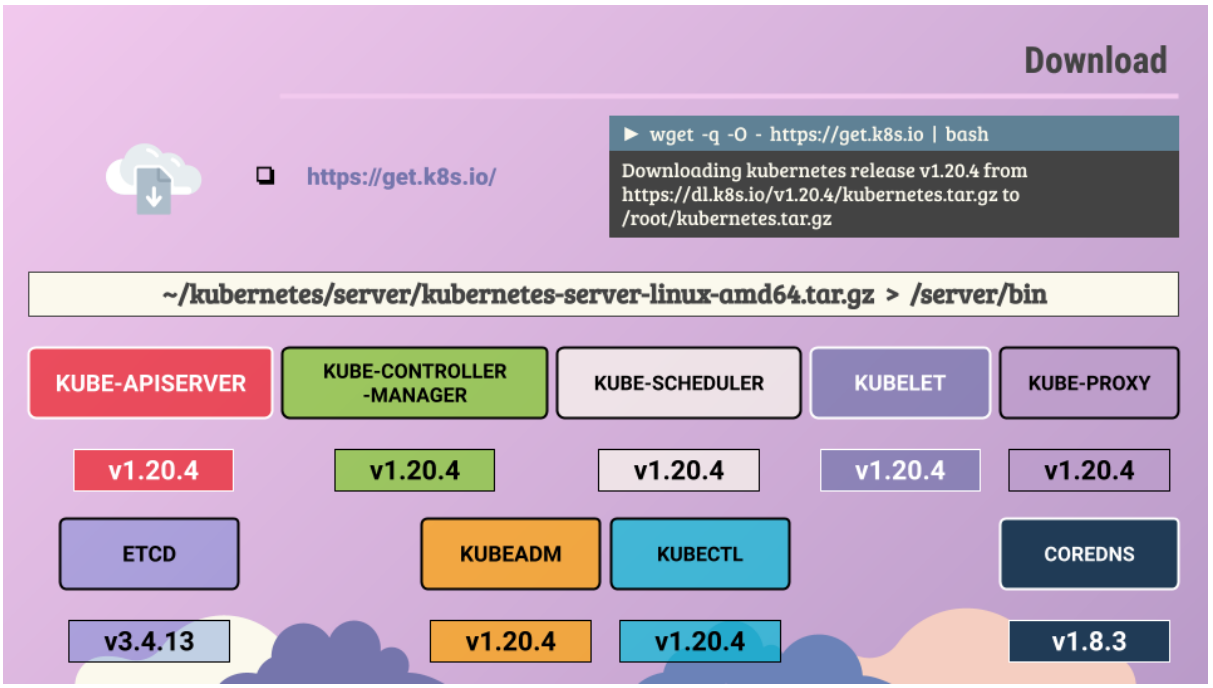
И вот, наконец, после этого они попадают в основной стабильный релиз.

Ты можешь найти все выпуски Kubernetes на странице релизов репозитория Kubernetes в Github.

Как получить последний релиз?

Загрузи файл kubernetes.tar.gz из репозитория или сделай еще проще - скачай скрипт с `https://get.k8s.io`.

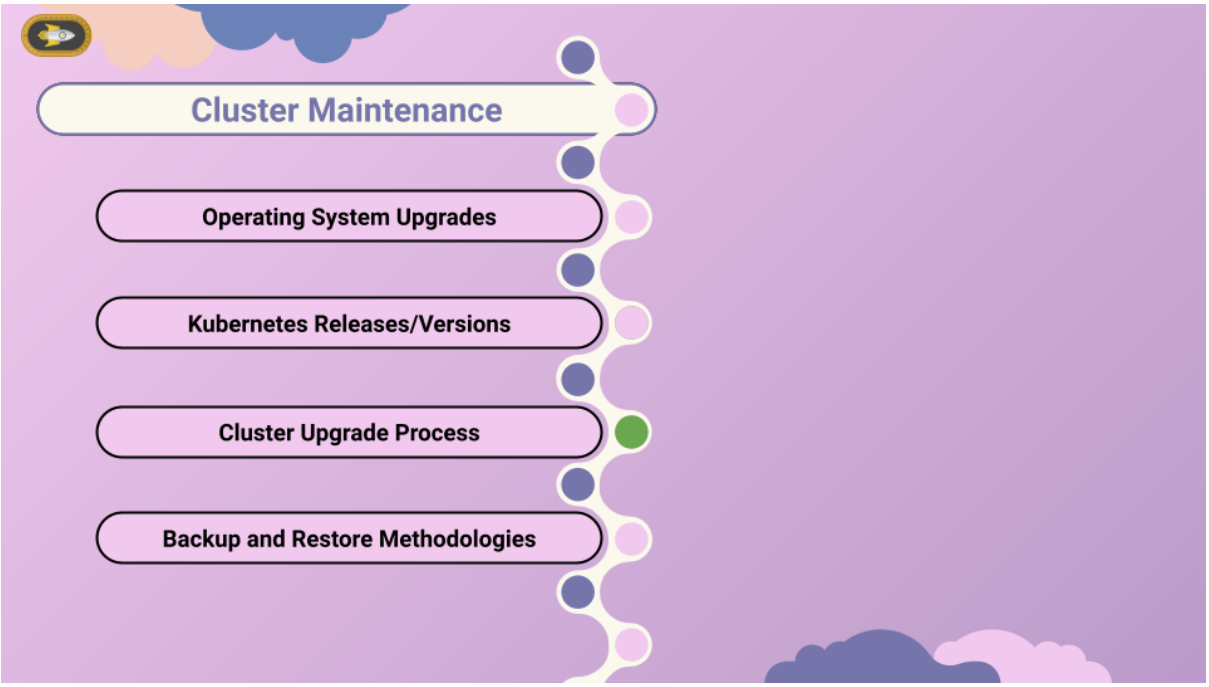
Распакуй его, и там ты найдешь исполняемые файлы для всех компонентов Kubernetes. Этот архив содержит все уже скомпилированные компоненты controlplane. Также там есть утилита управления и сетапа кластера. Все они одной версии.



Помни, что в controlplane есть другие компоненты, не относящиеся напрямую к проекту Kubernetes. У них будут другие номера версий. Кластер ETCD и сервер CoreDNS имеют свои версии, поскольку являются отдельными проектами.

Обычно примечания к выпуску каждого релиза предоставляют информацию о поддерживаемых версиях внешне зависимых приложений, таких как ETCD, CoreDNS и т. д.

Это был краткий обзор релизов и версий. В следующей лекции мы обсудим, как перейти с одной версии на другую.



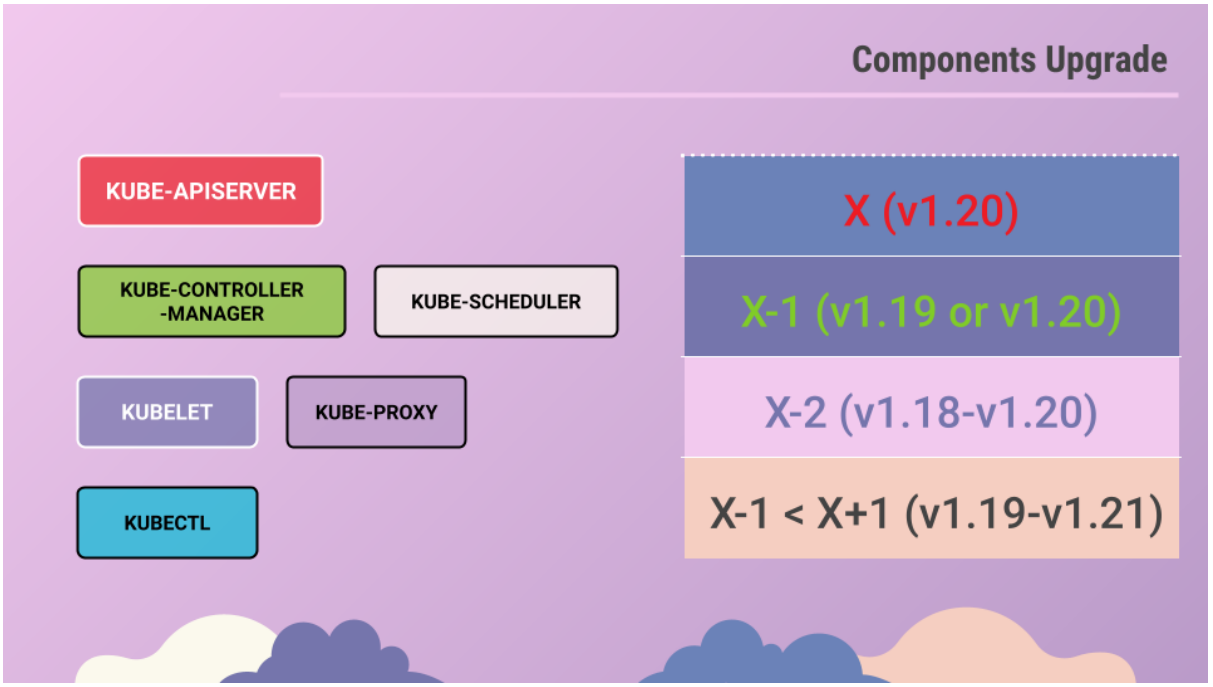
Привет и добро пожаловать на лекцию.
В этой лекции мы обсуждаем процесс обновления кластера в Kubernetes.

В предыдущей лекции мы увидели, как Kubernetes управляет выпусками своего программного обеспечения и как разные компоненты имеют свои версии. Мы пока оставим зависимость от внешних компонентов, таких как ETCD и CoreDNS, и сосредоточимся на основных компонентах controlplane.

Обязательно ли, чтобы все они имели одну и ту же версию?
Нет. Компоненты могут быть разных релизных версий. Поскольку kube-apiserver является основным компонентом в плоскости управления, и это компонент, с которым все взаимодействуют, все другие компоненты не должены иметь версию выше, чем у kube-apiserver. Контроллер-менеджер и планировщик могут быть на одну версию ниже.

Если мы примем версию kube-apiserver за X, то kube-controller-manager и kube-scheduler могут быть в версии X-1, а компоненты kubelet и kube-proxy могут быть на две версии ниже, т.е. X-2.

Т.е. если бы kube-apiserver был версии 1.20, kube-controller-manager и kube-scheduler могли бы быть версий 1.20 или 1.19, а kubelet и kube-proxy - еще и 1.18. Как я говорил, ни один из них не может быть версии выше, чем kube-apiserver, например версии 1.21.



Но это не относится к kubectl. Утилита kubectl может иметь версию 1.21, т.е. выше, чем apiserver. Также у kubectl может быть версия 1.20, это та же версия, как и у сервера и 1.19 версия ниже, чем у apiserver.

Этот допустимый перекос в версиях позволяет нам выполнять обновления в реальном времени. При необходимости мы можем обновлять компонент за компонентом.

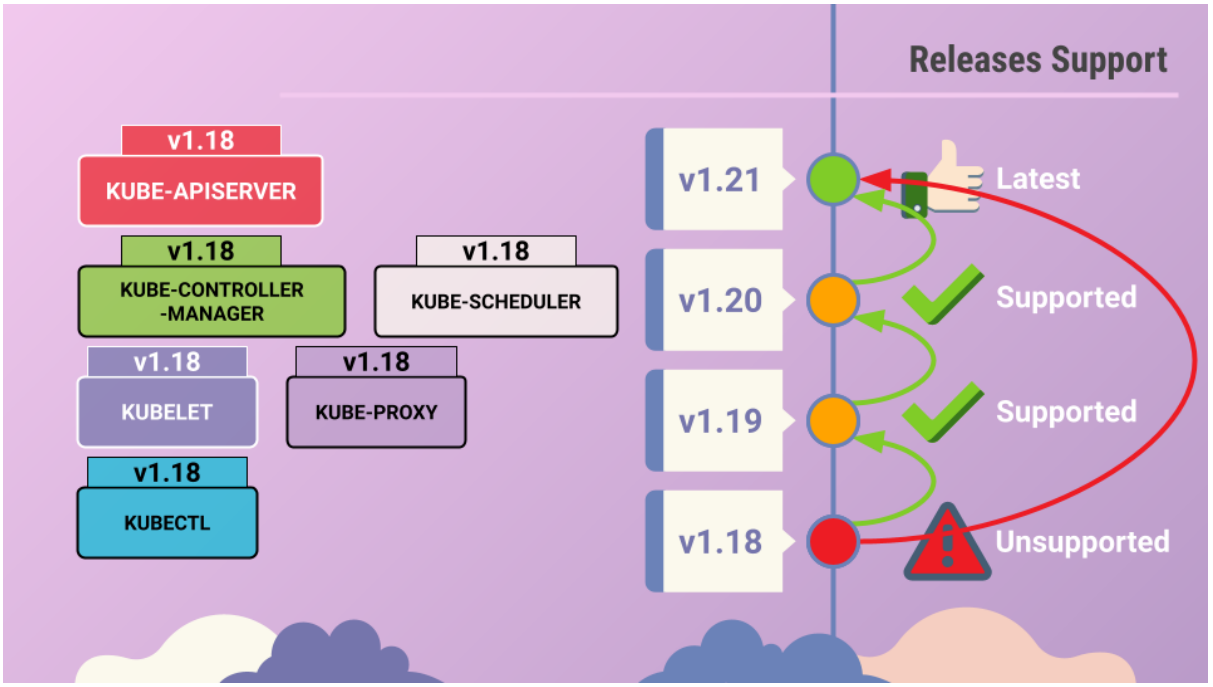
Когда нужно обновляться?

Допустим, у нас была версия 1.18, а Kubernetes выпускает версии 1.19 и 1.20.
В любой момент Kubernetes поддерживает только до трех последних минорных версий.

Поскольку 1.20 является последним выпуском, Kubernetes поддерживает версии 1.20, 1.19 и 1.18.

При выпуске 1.21 будут поддерживаться только версии 1.21, 1.20 и 1.19.
Перед выпуском 1.21 самое время обновить кластер до следующего выпуска, т.к. 1.18 уже не поддерживается.

Как мы обновляем? Мы обновляем напрямую с 1.18 до 1.21? Нет.



Рекомендуемый подход - обновлять по одной минорной версии за раз:
- версию с 1.18 до 1.19
- затем с 1.19 по 1.20
- а после с 1.20 по 1.21

Процесс обновления зависит от того, как настроен твой кластер.
Например, если он представляет собой управляемый кластер Kubernetes, развернутый у клауд-вендоров, таких как Google, то Google Kubernetes Engine позволяет легко обновить кластер всего за несколько щелчков мышью.

Если ты развернул кластер с помощью таких инструментов, как kubeadm, этот инструмент может помочь тебе спланировать и обновить кластер.

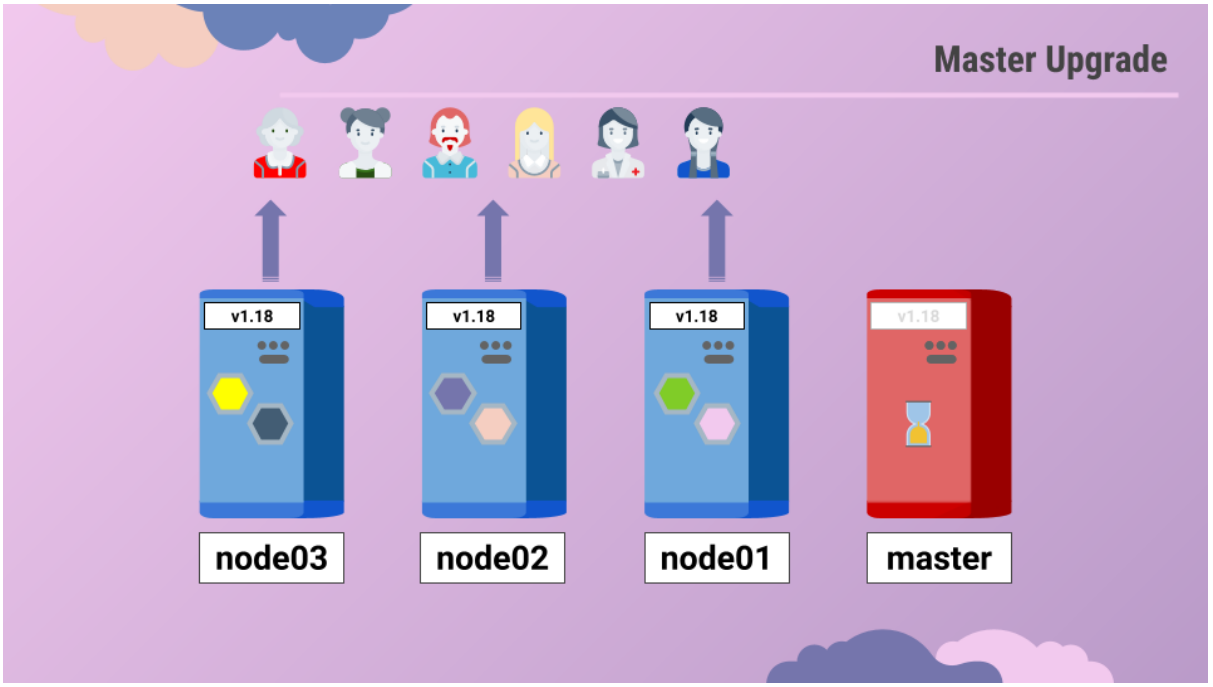
Если же ты развернул кластер с нуля, тебе вручную придется обновлять различные компоненты кластера. В этой лекции мы рассмотрим варианты обновления с kubeadm.

Итак, у нас есть кластер с главным и рабочим узлами, работающими в производственной среде, размещающими PODs и обслуживающими пользователей.

Узлы и компоненты имеют версию 1.18.

Обновление кластера состоит из двух основных этапов.
Сначала мы обновляем свои мастера, а затем обновляем рабочие узлы.

Во время обновления мастера компоненты controlplane, такие как apiserver, планировщик и менеджер контроллеров, ненадолго отключаются.



Но вывод из эксплуатации мастера не означает, что наши рабочие узлы и приложения в кластере затронуты. Все рабочие нагрузки, размещенные на worker-нодах, продолжат обслуживать пользователей в обычном режиме.

Поскольку мастер не готов, все функции управления отключены.
Ты не сможешь получить доступ к кластеру с помощью kubectl или другого API Kubernetes. Ты не сможешь развертывать новые приложения, удалять или изменять существующие.

Контроллер-менеджер тоже не работает. Следовательно если POD вышел из строя, то новый POD не будет создан автоматически, чтобы заменить старый.

Пока работают узлы и PODs, наши приложения будут работать, и пользователи не пострадают. После завершения обновления мастер будет забекаплен. Потом он вернется и раздаст указания, если где-то что-то нужно поправить. Вновь возвратится нормальная рабочая ситуация.

Ок, теперь у нас есть мастер с controlplane версии 1.19 на борту, но рабочие узлы все еще версии 1.18.
Как мы видели ранее, это все еще официально поддерживаемая версия, и она не выше версии kube-apiserver, значит все в порядке.
Пришло время обновить рабочие узлы.
Существуют различные стратегии обновления рабочих нод.

Первая из них - обновить все сразу, но тогда наши PODs выключаться, и пользователи больше не смогут получить доступ к приложениям.

После завершения обновления выполняется резервное копирование узлов, новые PODs назначаются, и пользователи могут возобновить доступ. Это единственная стратегия, приводящая к простою.

Вторая стратегия - обновлять по одному узлу за раз.

Возвращаемся к состоянию, когда у нас есть обновленная мастер-нода и ноды, ожидающие обновления.

Сначала мы обновляем первую ноду. Рабочие нагрузки с нее перемещаются на второй и третий узел, и пользователи соответственно ходят туда же.

После обновления и резервного копирования первого узла мы обновляем вторую ноду, рабочие нагрузки которой перемещаются на первую и третью ноды.

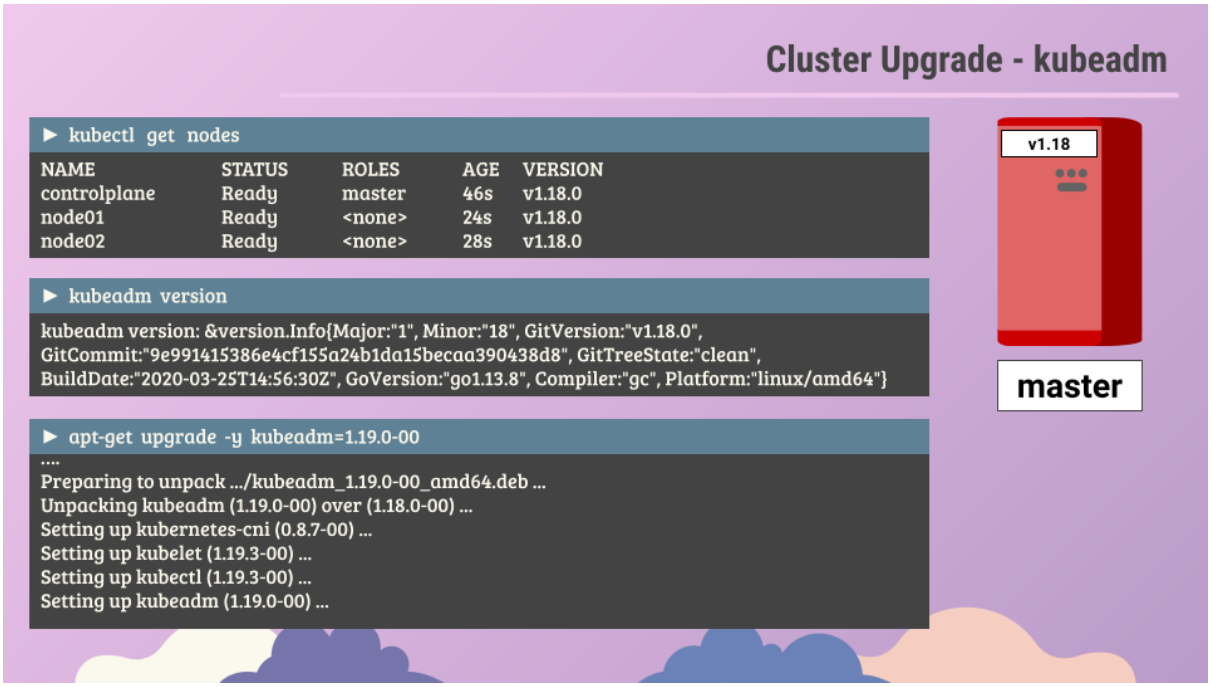
Наконец, третий узел, все по тому же сценарию, его рабочие нагрузки распределяются между первыми двумя.

Мы следуем той же процедуре, пока мы не обновим все узлы до более новой версии, следующий этап обновить ноды с версии 1.19 до 1.20, а затем до 1.21.

Третья стратегия - добавление новых узлов в кластер.
Эти узлы уже с более новой версией программного обеспечения.
Это особенно удобно, если мы работаем в облачной среде, где можно легко подготовить новые ноды и удалить старые.

Итак мы присоединяем новую ноду с более высокой версией программного обеспечения и перемещаем рабочую нагрузку на нее. Далее удалим старый узел без нагрузки. Делаем так, пока у нас, наконец, все ноды не станут с новой версией ПО.

Давай теперь посмотрим, как это делается руками.
Скажем, у нас кластер из двух воркеров и мастера и мы должны обновить свой кластер с 1.18 до 1.19



Запустим команды и узнаем версии компонентов - master, workers и kubeadm версии 1.18.

Теперь я обновлю kubeadm до версии 1.19. Через apt-get я получил версию kubeadm 19.3.
Также вместе с ним он обновил kubelet до версии 1.19.3.

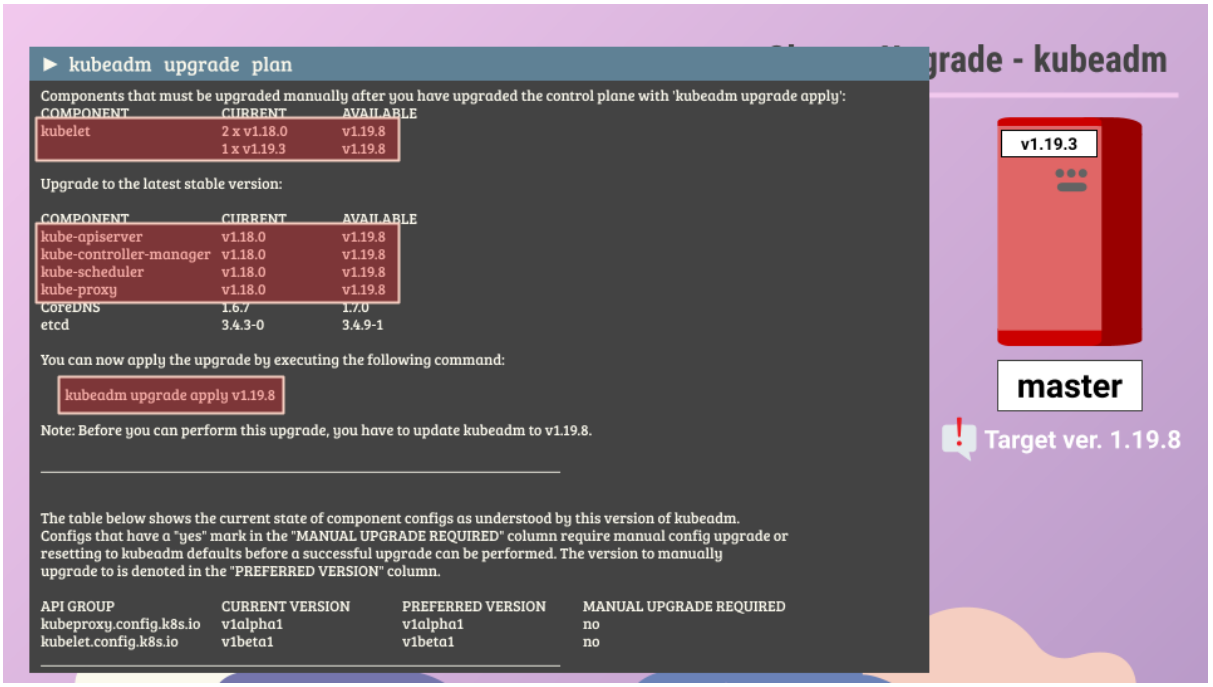
Делаем kubectl get nodes и видим v.1.19.3. Дело в том, что результат этой команды дает не актуальную версию controlplane, а всего лишь показывает версию kubelet, установленного на узле.

В kubeadm есть команда `upgrade`, которая помогает обновлять кластеры.

С помощью kubeadm запусти команду `kubeadm upgrade plan`, и она предоставит тебе много полезной информации. Текущая версия кластера, версия инструмента kubeadm, последняя стабильная версия Kubernetes.

Затем в ней перечислены все controlplane компоненты, их версии, а также версия, до которой они могут быть обновлены.

Он также сообщит тебе, что после обновления компонентов уровня управления ты должен вручную обновить версии kubelet на каждом узле.
Как ты помнишь kubeadm не устанавливает и не обновляет kubelets.



Наконец, он даст тебе команду для обновления кластера.
У меня это kubeadm upgrade apply v1.19.8

Итак, моя целевая версия будет v1.19.8.

Также обрати внимание, что перед обновлением кластера тебе потребуется обновить саму утилиту kubeadm до этой целевой версии.

Обновим кэш apt.

У нас уже kubeadm 1.19.3, и мы хотим перейти на 1.19.8, но как ты видел, нам доступна также версия 1.20.4. Ты помнишь, что мы можем использовать только одну младшую версию за раз при обновлении кластера, поэтому мы не обновляем kubeadm до 1.20.

Теперь обновлю сам инструмент kubeadm до версии 1.19.8 командой
apt-get install -y kubeadm=1.19.8-00

Затем обновим кластер, используя команду из выходных данных команды upgrade plan, а именно kubeadm upgrade apply v1.19.8

Kubeadm пуллит необходимые образы и обновляет компоненты кластера.
После завершения наши компоненты уровня управления теперь находятся на уровне 1.19.

Если ты запустишь команду kubectl get nodes, ты все равно увидишь версию мастера 19.3.
Почему так, ты наверное догадываешься. Да, это версия kubelet с мастер-узла, зарегистрированного в apiserver.

Поэтому следующий шаг - обновление kubelet.

Как ты помнишь, что от вида сетапа зависит есть ли на мастере kubelet или нет.

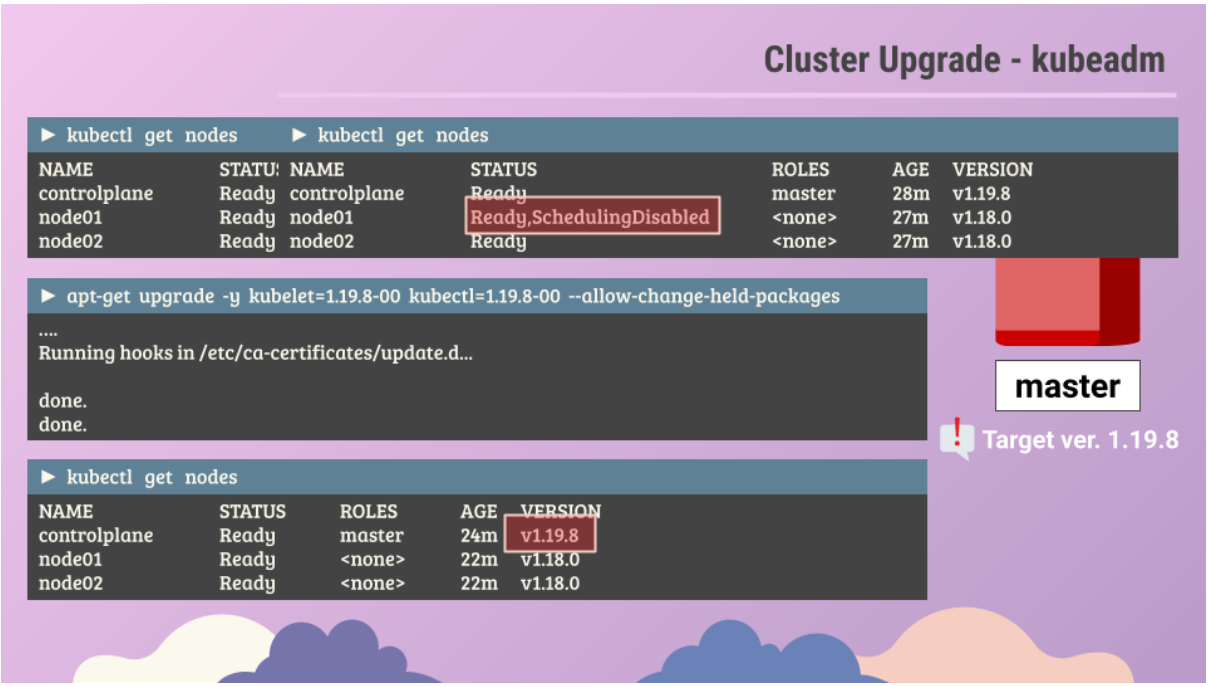
В этом случае кластер развернут с помощью kubeadm и он имеет kubelet на главном узле. Kubelet на мастере запускает контейнеры компонентов controlplane.

Когда мы позже в ходе этого курса будем настраивать кластер Kubernetes с нуля, мы не будем ставить kubelet на мастер-узел. И в том сетапе у этой команды не будет мастера в выводе.

Ок, как я сказал, следующим нашим шагом будет обновление kubelet на главном узле, если на нем нужен kubelet.

Для этого выполни команду `apt-get upgrade -y kubelet=1.19.8-00 kubectl=1.19.8-00 --allow-change-held-packages`.

После обновления пакета перезапусти службу kubelet, если он сам этого не сделал.



Выполнение команды kubectl get nodes теперь показывает, что мастер обновлен до 1.19.8, а рабочие узлы по-прежнему находятся на уровне 1.18.

Двигаемся к воркер-нодам.

Наша стратегия - обновлять по одному. Сначала нужно переместить рабочие нагрузки с первого рабочего узла на другой узел.

Команда kubectl drain позволяет безопасно завершить работу всех PODs на одном узле и переназначить их на другие.

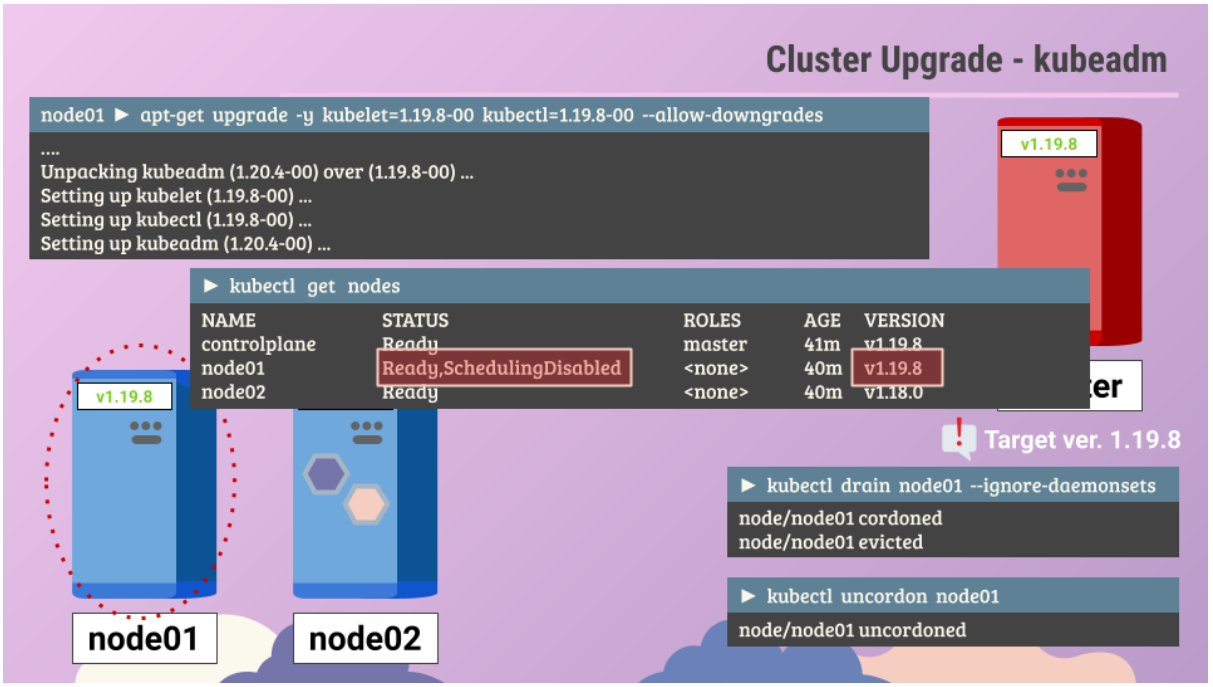
Он также делает `cordon` на узел, т.е. отмечает, что он не подлежит планированию. Таким образом, на нем не будет запланировано никаких новых PODs, а старые уйдут.

Ты можешь убедится в этом в поле status команды get nodes.

Затем обновим пакеты kubeadm и kubelet на рабочих узлах, как мы это сделали на главном узле. Сделаем kubectl get nodes. Опять версия kubelet играет с нами шутки. Теперь она уже стала 1.20. Как ты помнишь, kubelet не может обгонять версию kubeapi, т.е. быть версии 1.20. Нам нужно будет ее понизить.

Сейчас с помощью команды kubeadm upgrade node обновим конфигурацию узла для новой версии kubelet и перезапустим службу kubelet.

Сделаем downgrade командой apt-get upgrade -y kubelet=1.19.8-00 kubectl=1.19.8-00 --allow-downgrades



Теперь на узле должна быть установлена новая и нужная нам версия программного обеспечения. Однако, когда мы делали drain на узел, мы фактически пометили его как неназначаемый. Нам нужно это снять, выполнив команду kubectl uncordon node01.

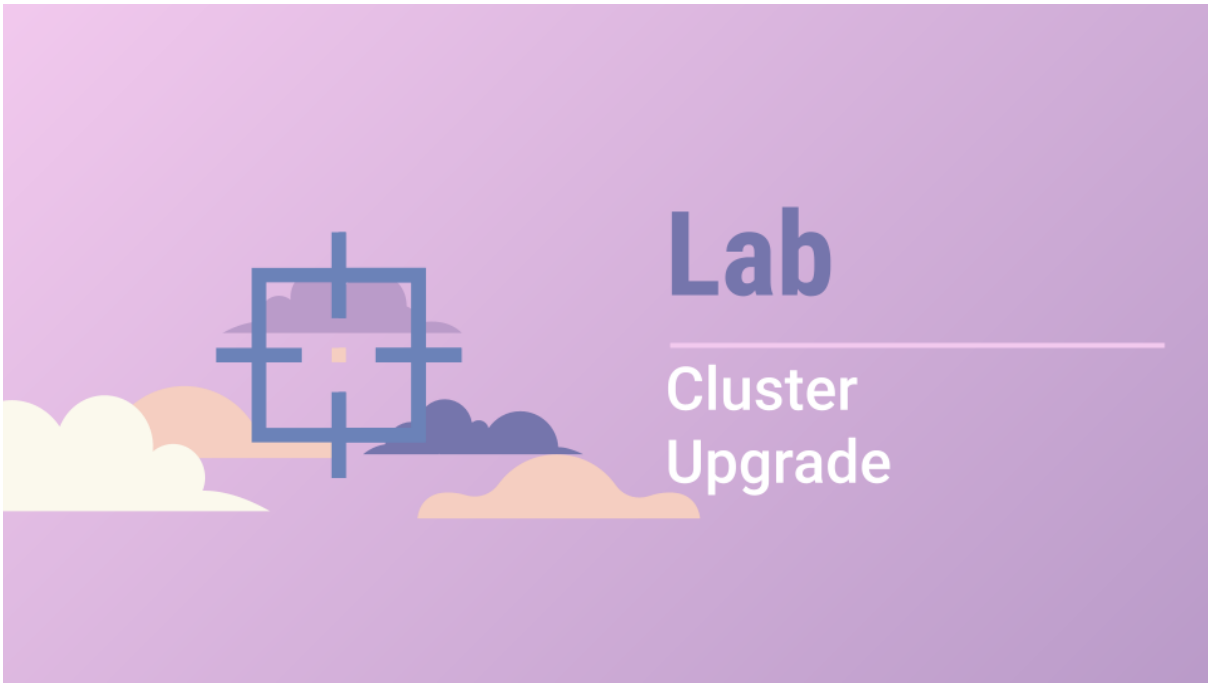
Ок, этот узел доступен к планировке, но помни, что совсем не обязательно, что PODs сами вернуться на этот узел. Он только теперь отмечен как планируемый. Только когда PODs будут удалены из других узлов или когда будут запланированы новые PODs, этот узел примет рабочую нагрузку.

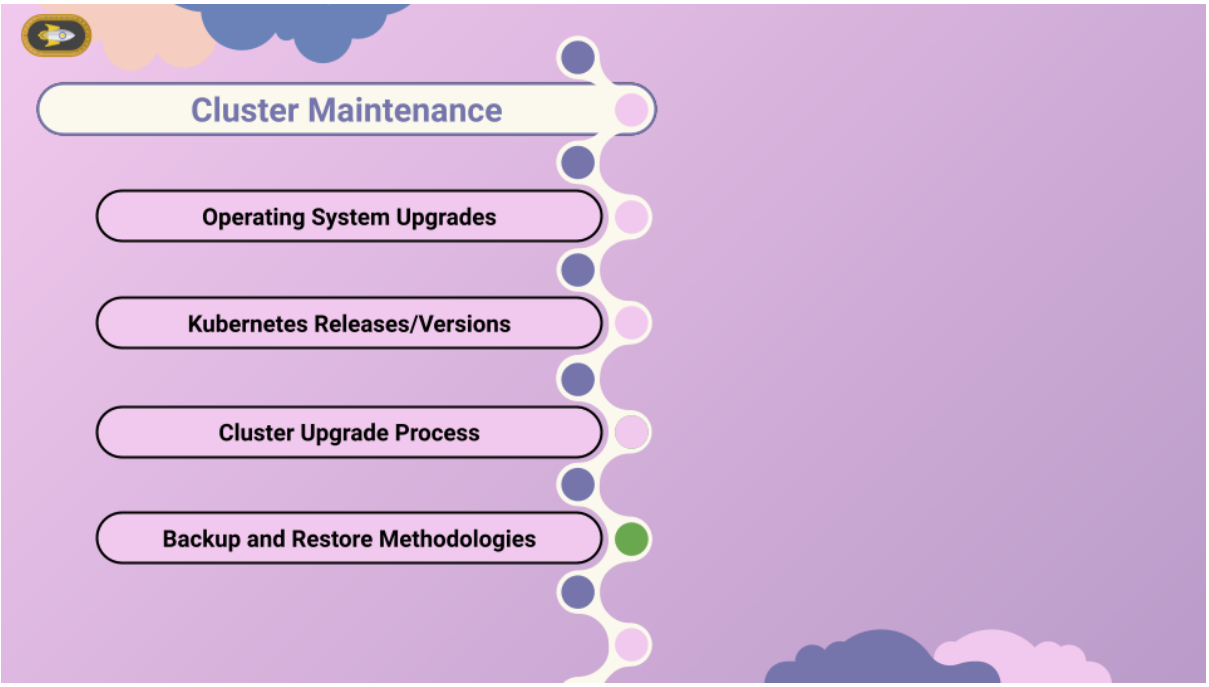
И это скоро произойдет, ведь мы будем выводить из эксплуатации вторую ноду, чтобы выполнить те же действия по ее обновлению.

Как видишь, теперь у нас полностью обновлённый до версии 1.19 кластер, и мы сделали это бесшовно.

Это все в этой лекции. Перейди к практическому тесту, где ты потренируешься в обновлении живого кластера с запущенными в нем приложениями, не останавливая их.

Удачи.


Lab
Cluster
Upgrade



Привет и добро пожаловать на лекцию. В этой лекции мы рассмотрим различные методологии резервного копирования и восстановления.

Давай начнем с того, что порассуждаем о резервном копировании в кластере Kubernetes.

До этого времени в курсе, мы разворачивали ряд различных приложений в нашем кластере Kubernetes с использованием файлов определений services, deployments и PODs. Неплохо бы держать их копии в резерве. Мы знаем, что в базе ETCD хранится вся информация, относящаяся к кластеру. Это важно.

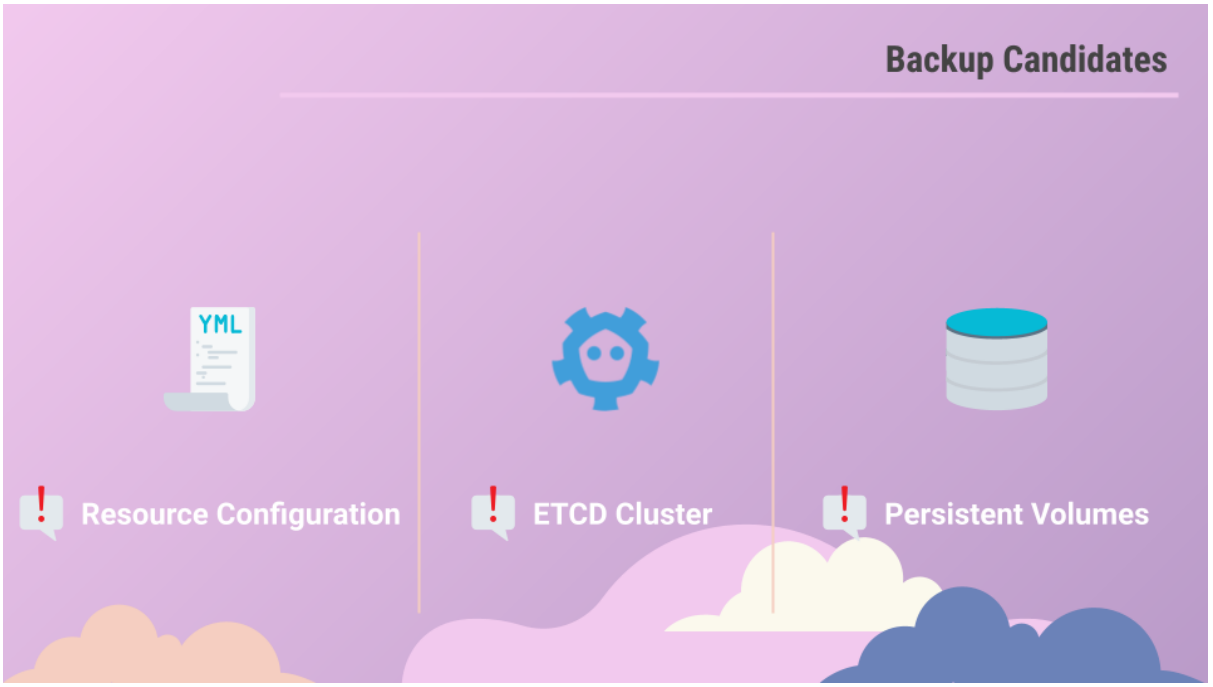
И еще, если наши приложения настроены на постоянное хранилище, это тоже кандидат на резервное копирование.

Что насчет ресурсов, которые мы создали в кластере, когда мы использовали императивный способ создания объекта через выполнение прямой команды?

Например, при создании namespace, или secret, или configmap, или те случаи когда мы делали expose приложению, порождая для него службу в кластере.

С точки зрения сохранения конфигурации, декларативный подход, когда мы сначала создавали файл определения, а затем выполняли команду kubectl apply для этого файла будет более предпочтительным.

С этим подходом у нас останутся все объекты, необходимые функционирования приложения, в виде файлов определения в одной папке. Их можно легко использовать повторно в более позднее время или поделиться с другими. Конечно, мы должны иметь копию этих файлов в резервном хранилище.



Хорошая практика - хранить их в репозиториях исходного кода. Таким образом, наша команда сможет поддерживать репозиторий исходного кода с соответствующими решениями для резервного копирования.

С управляемыми/общедоступными репозиториями исходного кода, такими как Github, нам не о чем беспокоиться. При этом, даже если мы вдруг совсем потеряем кластер, то сможем повторно развернуть приложение, просто применив сохраненные файлы конфигурации. Хотя декларативный подход является предпочтительным, необязательно, чтобы все члены нашей команды будут придерживаться этих стандартов.

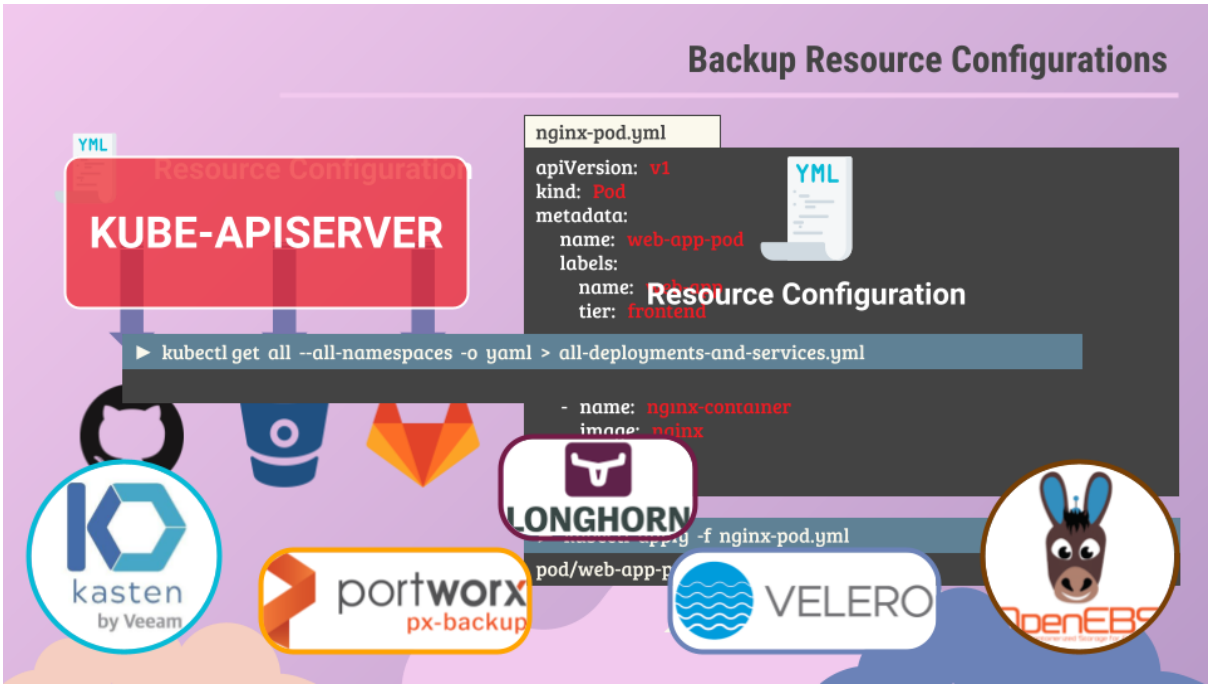
Итак, что, если кто-то создал объект императивным способом, нигде не документируя эту информацию? Мы просто не найдем этот объект.

Таким образом, лучший подход к резервному копированию конфигурации ресурсов - использовать запросы к серверу kube-api.

Выполним запрос к API-серверу с помощью kubectl или путем прямого доступа к через API и сохраним все конфигурации ресурсов. Для всех объектов, созданных в кластере, там присутствуют их описательные двойники.
Например, одна из команд, которые можно использовать в сценарии резервного копирования.

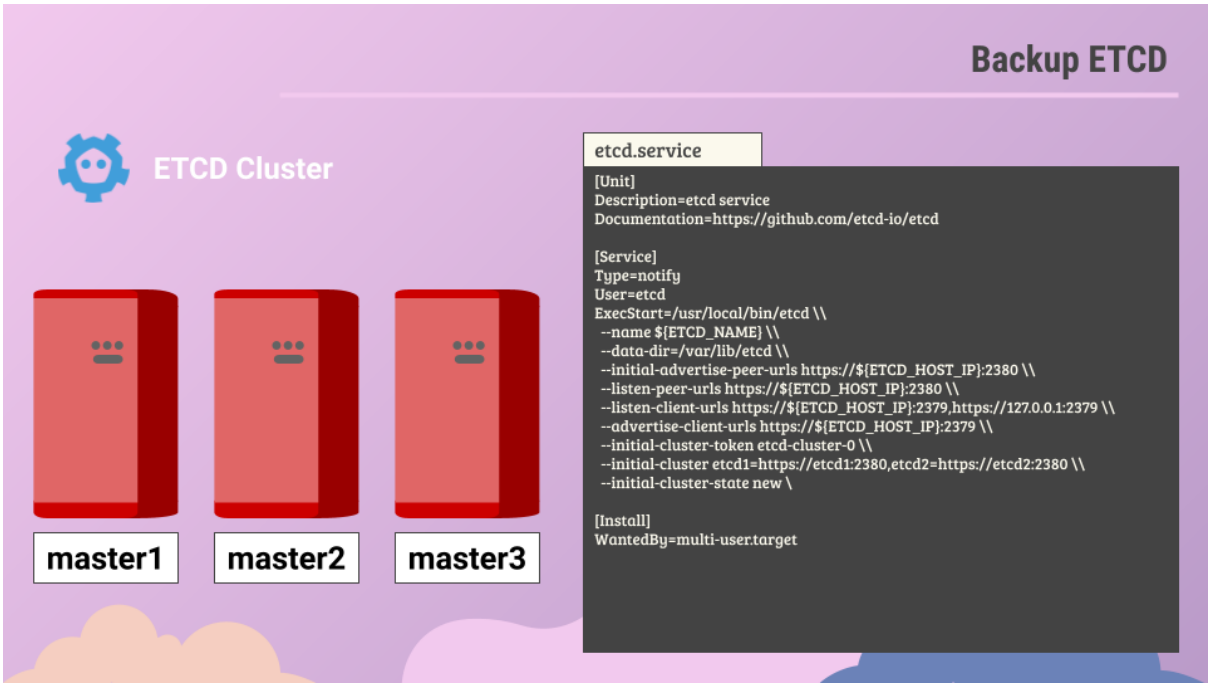
kubectl get all --all-namespaces -o yaml

Она позволяет получить все PODs, deployments и services во всех пространствах имен, а данные из вывода команды мы сохраним в виде YAML-манифеста.
Но это только несколько групп ресурсов. Необходимо учитывать множество других групп.



Конечно, нам не нужно разрабатывать эти решения самостоятельно. Есть такие инструменты, как Velero (бывший Heptio Ark), Kasten, Longhorn и т.д., которые могут сделать это за нас. Точнее помогут нам в создании резервных копий нашего кластера Kubernetes с помощью Kubernetes API.

Теперь перейдем к ETCD. Кластер ETCD хранит информацию о всем состоянии нашего кластера. Таким образом, здесь хранится информация о самом кластере, узлах и всех других ресурсах, созданных в кластере. Поэтому вместо резервного копирования ресурсов, как мы обсуждали раньше, мы можем выбрать бекап самого сервера ETCD.



Как мы видели, в большинстве случаев кластер ETCD размещается на мастер-узлах. При конфигурировании ETCD мы указали место, где будут храниться все данные. Каталог данных. Это каталог, который можно настроить для резервного копирования с помощью решения для резервного копирования. ETCD также поставляется с подобным встроенным решением для создания снэпшотов.

Мы можем сделать snapshot базы данных ETCD с помощью команды

etcdctl snapshot save snapshot.db

Здесь snapshot.db - имя файла для сохранения. Этот файл снэпшота создается в текущем каталоге. Если требуется создать в другом месте, можно просто указать полный путь.

Теперь мы можем просмотреть состояние резервной копии с помощью команды

etcdctl snapshot status snapshot.db

Чтобы восстановить кластер из этой резервной копии позже нам потребуется:

- сначала остановить службу kube-apiserver, так как процесс восстановления потребует перезапуска кластера ETCD, а сервер kube-api зависит от него
- затем запустить команду `etcdctl  snapshot  restore snapshot.db`, указав путь к файлу резервной копии, который является файлом snapshot.db.

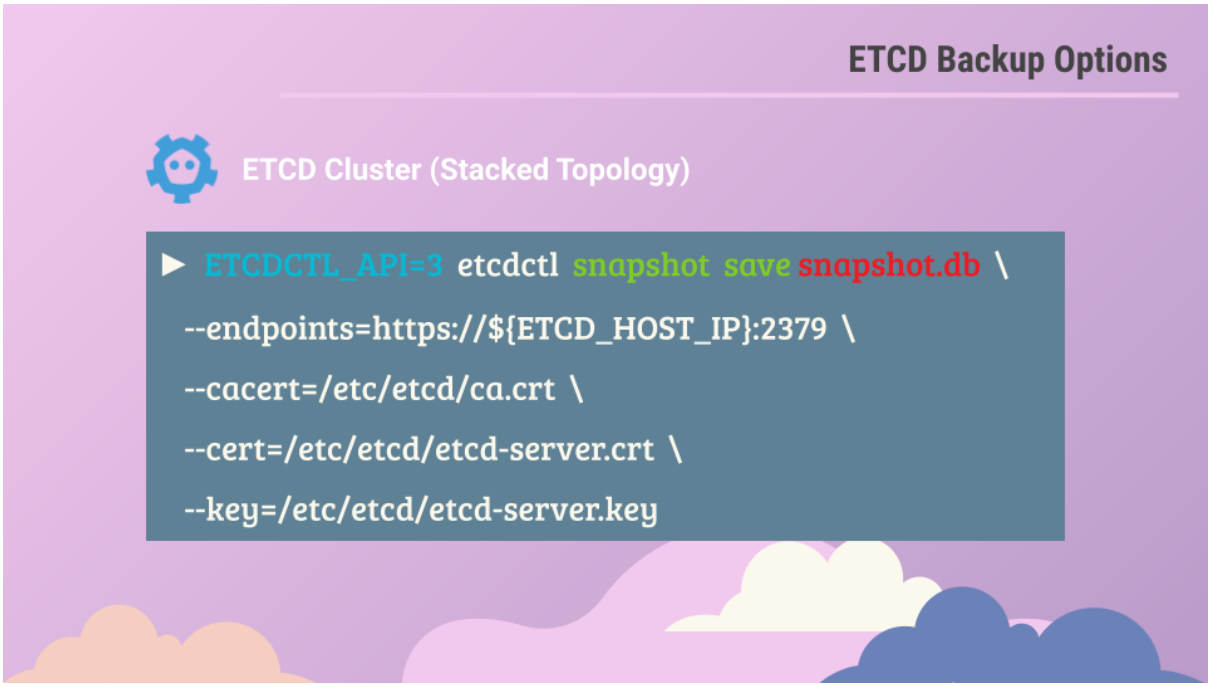
Когда ETCD восстанавливается из резервной копии, она инициализирует новую конфигурацию кластера и настраивает участников ETCD в качестве новых участников для нового кластера. Это сделано для предотвращения случайного присоединения нового члена к существующему кластеру.

Скажем, например, я использую этот снэпшот для подготовки нового кластера ETCD в целях тестирования. Я не хочу, чтобы участники нового тестового кластера случайно присоединились к производственному кластеру. Поэтому во время восстановления мне нужно указать новый токен кластера и те же параметры начальной конфигурации кластера, которые указаны в исходном файле конфигурации.

При запуске этой команды создается новый каталог данных. В этом примере в расположении `/var/lib/etcd-backup`.

- после мы настраиваем файл конфигурации службы ETCD для использования нового токена кластера и каталога данных.
- перезагрузим сервисный демон и перезапустим службу etcd.
- наконец, запустим службу kube-apiserver.

Теперь наш кластер должен вернуться в исходное состояние.



Несколько замечаний, прежде мы закончим.
В этом примере kube-apiserver находится на той же машине, что и ETCD, так называемая stacked topology. В других сетапах нужно будет поправить конфигурацию API-сервера, т.к. ETCD может быть уже на другом IP.
Со всеми командами ETCD не забудь указать файлы сертификатов для аутентификации. Укажи конечную точку для кластера ETCD и сертификат CA, сертификат etcd-server и ключ.
Еще прими во внимание, что у etcdctl версии 3.4+ тебе уже не нужно ставить префикс ETCDCTL_API=3 перед командой, в версиях ниже она выдаст ошибку.

Итак, мы рассмотрели два варианта: резервное копирование с использованием ETCD и резервное копирование путем запроса kube-apiserver. У обоих есть свои плюсы и минусы.

Если ты используешь управляемую среду Kubernetes, то у тебя просто может не быть доступа к кластеру ETCD, в этом случае резервное копирование путем запроса kube-apiserver, вероятно, лучший способ.

Ну вот и все в этой лекции. Перейди к практическому упражнения попробуй себя в резервном копировании ETCD, а затем восстановлении кластера из резервных копий.