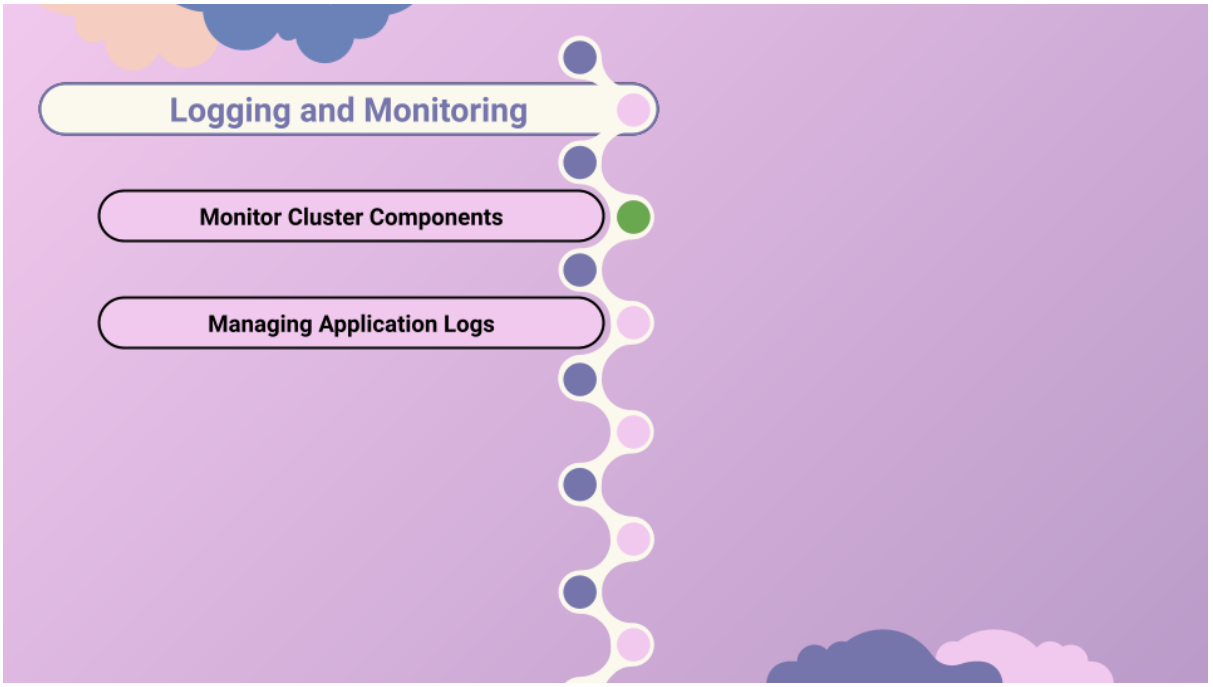


Привет и добро пожаловать на лекцию. В этой лекции мы говорим о мониторинге кластера Kubernetes.

Итак, как нам отслеживать потребление ресурсов в Kubernetes? Или, что более важно, что нам полезно бы было отслеживать?

Ну как минимум, хотелось бы знать, что там с инфраструктурой, т.е. показатели уровня ноды, такие как:

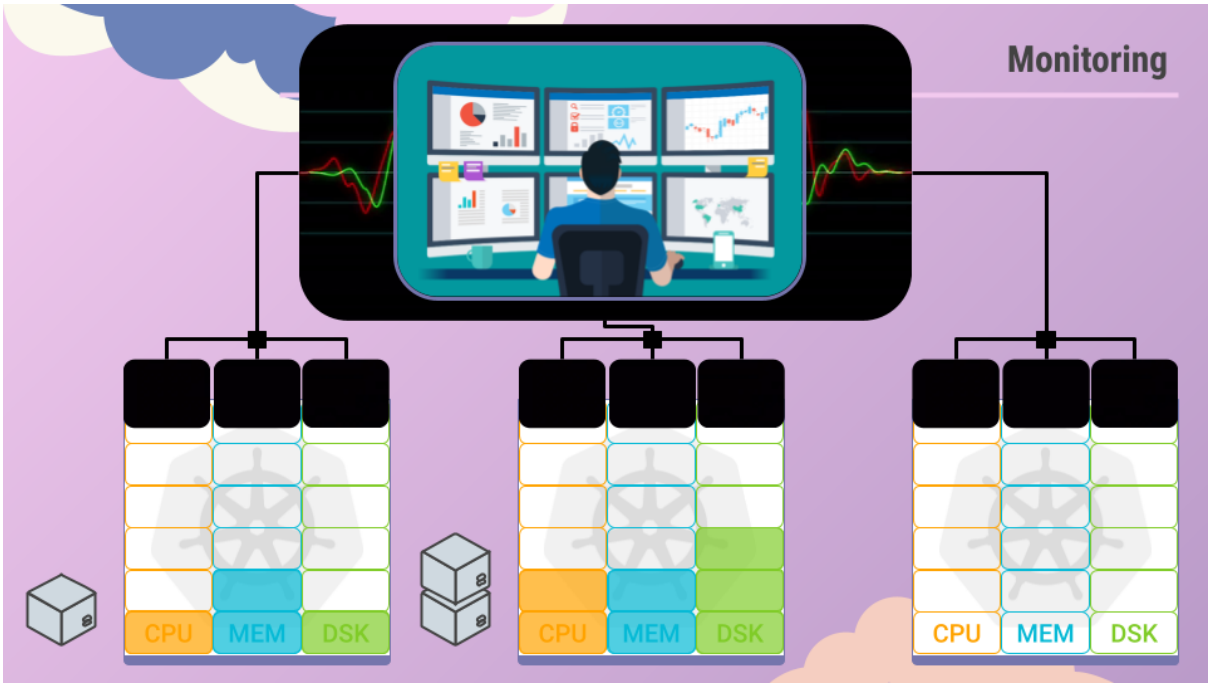
- количество узлов в кластере
- сколько из них исправно

, а также показатели производительности, такие как:

- использование CPU
- использование памяти
- место на дисках
- пропускная способность сети

Еще неплохо получить информацию уровня POD, такую как их количество, и метрики производительности каждого POD. Например такие, как потребление ими CPU и памяти.

Также, я бы хотел знать, что там с приложением внутри, не вышло ли оно из строя и отвечает ли пользователю.



Поэтому нам нужно решение, которое будет отслеживать эти метрики, хранить их и предоставлять аналитику на основе этих данных. На момент создания курса Kubernetes не поставляется с полнофункциональным встроенным решением для мониторинга.

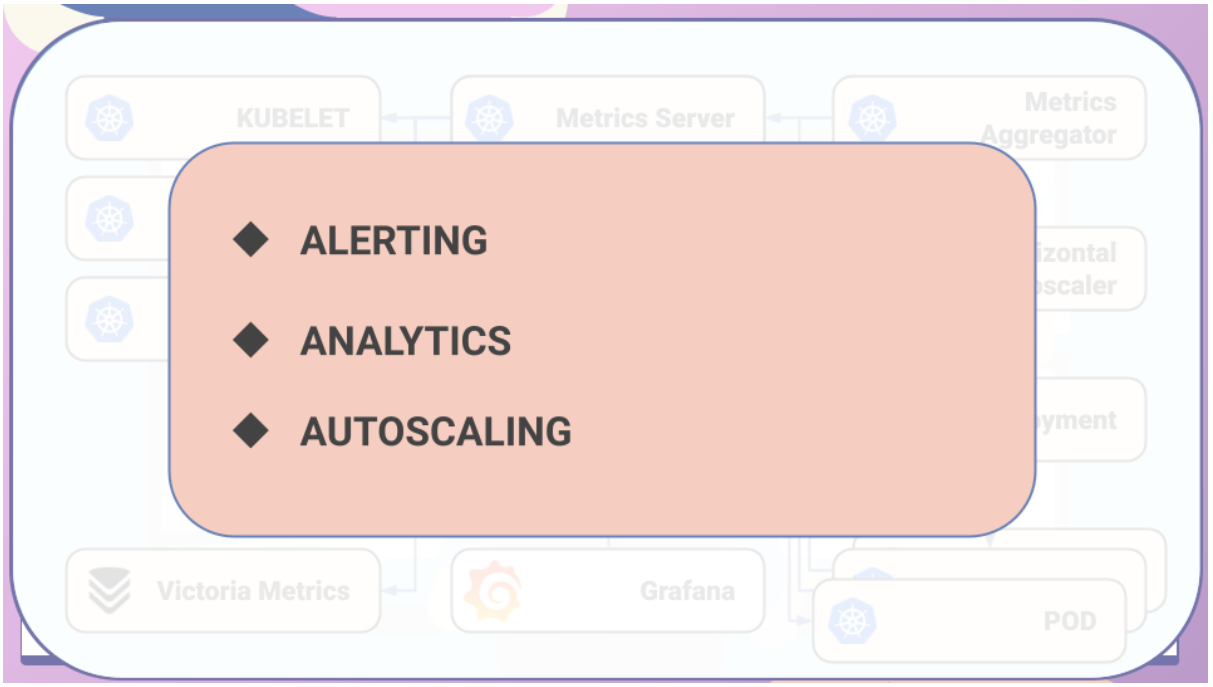
Тем не менее, фактическим стандартом для мониторинга стал Prometheus, этому способствует сообщество разработчиков Kubernetes, поддерживая интеграцию с ним практически `из коробки`.

Сегодня доступен ряд решений с открытым исходным кодом, таких как Metrics-Server, Prometheus, Elastic Stack, а также частично проприетарные решения, вроде Dynatrace или жестко проприетарные такие как Datadog.

Типовая схема мониторинга показана здесь, в силу динамичности кластера в нем действуют разные решения, обслуживающие разные области ответственности в части мониторинга.

Типичные функции это:

- сбор данных
- агрегация
- временное хранение
- долгосрочное хранение
- просмотр
- алертинг
- анализ



Итак на этой схеме у нас задействованы Prometheus, который снимает данные с приложений и показания нод, еще он заботится о выдаче данных для визуализации в Grafana, а также отправляет данные на хранение в Victoria Metrics.

Metrics Server работает в связке с Prometheus, и его задачей является быть источником информации для автоскейлера Kubernetes - horizontal pod autoscaler.

Когда ты будешь искать эталонные архитектуры для мониторинга Kubernetes, ты найдешь много справочных материалов в Интернете.

В общем, по задачам мониторинга я бы сказал что это три `A`:

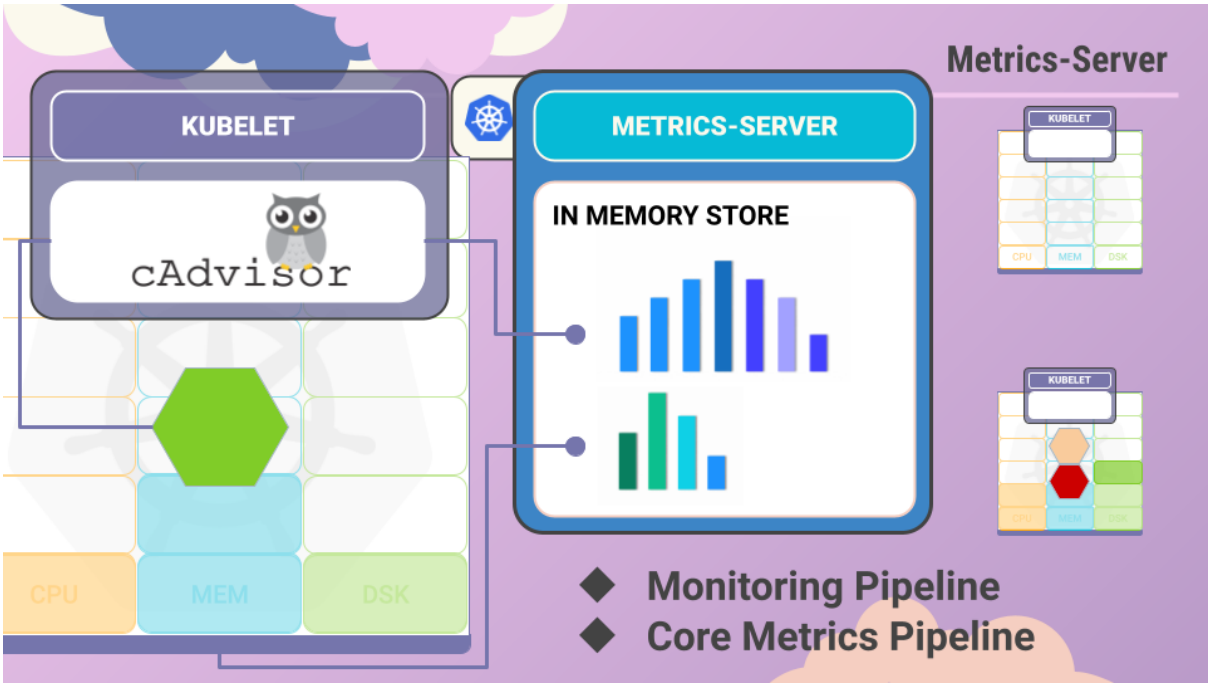
Alerting - ну тут понятно, что если что-то сломалось, мы бы об этом знали и смогли отреагировать.

Analytics - это возможность анализировать накопленный массив данных, и из него разные группы выносят свое:

- бизнес, где он зарабатывает, а где теряет
- админы и секьюрники - версии ПО и vulnerabilities
- SRE - соблюдение SLO и прочий UX

А еще мы там видим состояние нашей инфраструктуры в динамике, когда она вывозит, а когда нет, и где ее узкие места.

Это были концептуальные `A`
Третья `A` - что ни на есть самая прикладная для Kubernetes, т.к. от метрик CPU и памяти работает horizontal pod autoscaler, который управляет количеством реплик в deployment, тем самым обеспечивая работу под изменяющимся demand.



Внизу я дал ссылку на то, как устраивать инструментизацию, посмотри, это будет полезно.

Heapster был одним из первых проектов, в котором были реализованы функции мониторинга и анализа для Kubernetes.

Однако теперь Heapster устарел, и была создана упрощенная версия, известная как Metrics-Server.

У нас может быть один Metrics-Server на кластер. Metrics-Server получает метрики от каждого из узлов и PODs, агрегирует их и сохраняет в памяти.

Обрати внимание, что сервер метрик - это только решение для мониторинга in memory. Он не хранит метрики в постоянном хранилище.
В результате мы можем видеть лишь очень короткий промежуток данных, без возможности получить исторические данные о производительности.

Для этого мы должны положиться на одно из более крутых решений для мониторинга, о которых мы говорили чуть ранее в этой лекции.

Так как же генерируются метрики для PODs и nodes?

Существуют два источника. Первый известен как Monitoring Pipeline, это данные о CPU и памяти ноды.
Второй источник привлекает для работы компонент Kubernetes, который запущен в качестве агента на каждом узле, ты его знаешь, это kubelet.

У kubelet есть подкомпонент, известный как cAdvisor или Container Advisor. cAdvisor отвечает за получение метрик производительности из POD и отправку их через API

kubelet. Таким образом эти метрики становятся доступными для сервера метрик. Этот источник называется Core Metrics Pipeline.

Как нам это запустить?

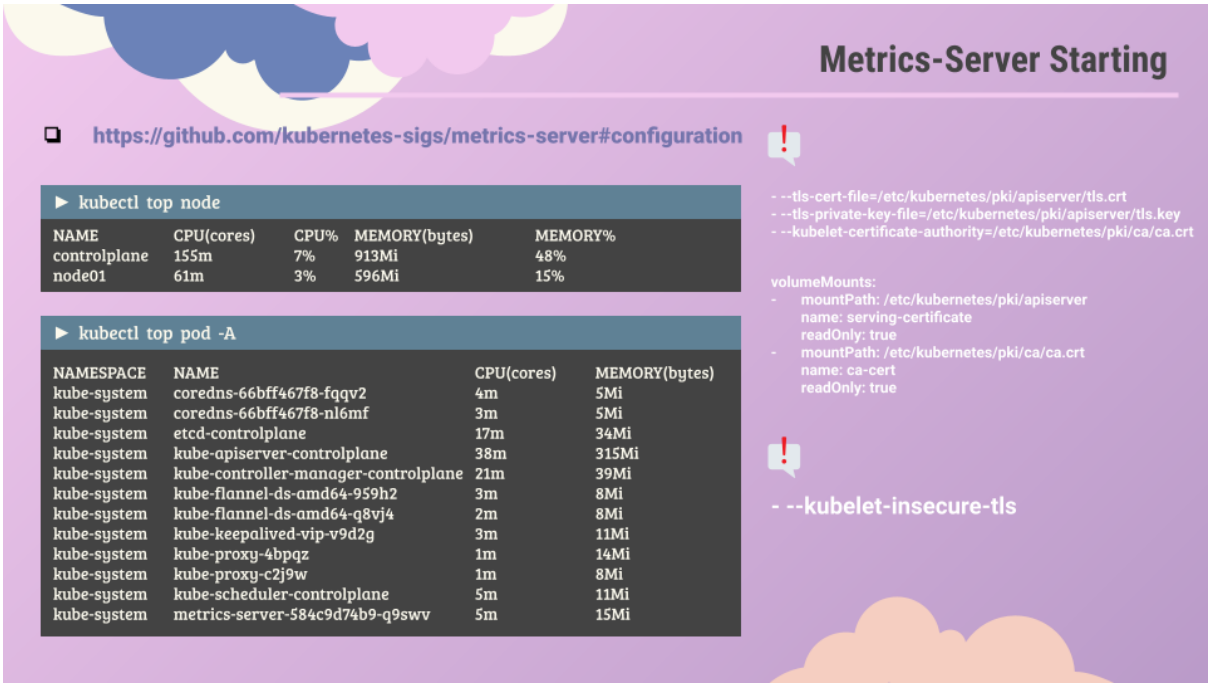
Если ты используешь minikube на своем локальном кластере, выполни команду minikube addons enable metrics-server.

Для всех остальных сред разверни Metrics-Server путем запуска манифеста, как ты видишь здесь. Эта команда развертывает deployment, services, roles, чтобы Metrics-Server смог запрашивать метрики производительности с узлов в кластере.

После развертывания дай серверу метрик некоторое время для сбора и обработки данных.

Производительность кластера можно увидеть, выполнив команду kubectl top node.

Это покажет потребление ресурсов CPU и памяти для каждого из узлов.



Как видишь, на моем мастере используется 7% ЦП, что составляет около 155 миллиядер.

Используй команду kubectl top pod для просмотра показателей производительности PODs.

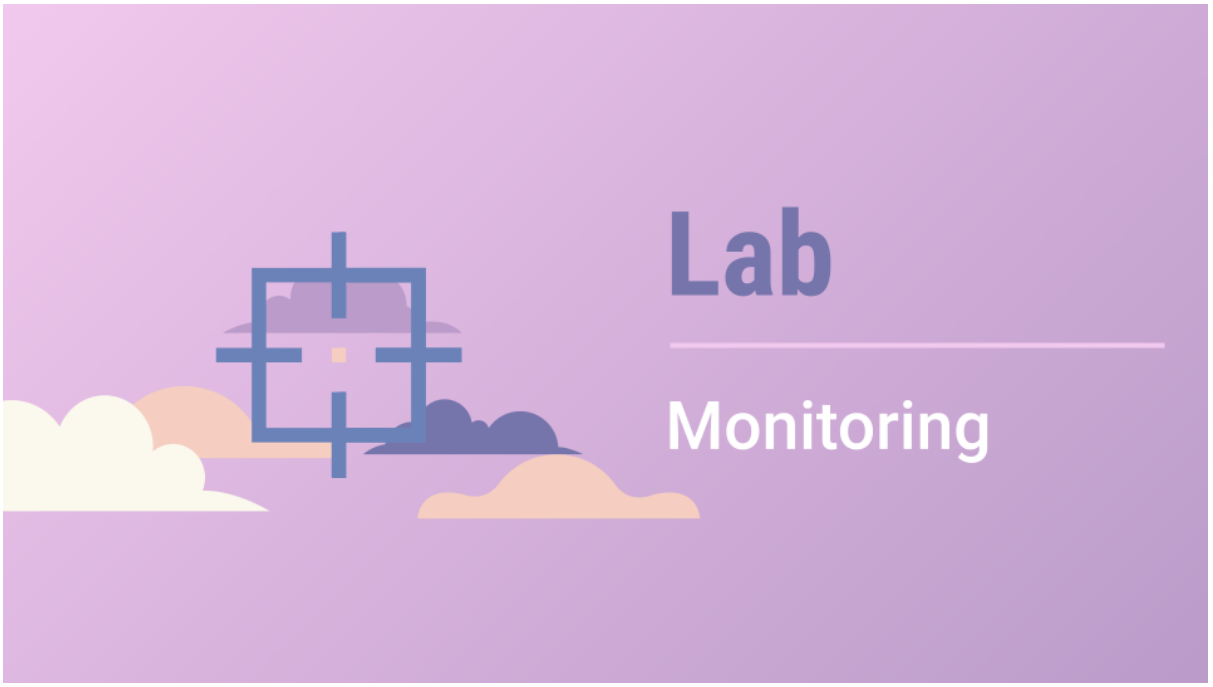
Если ты видишь, что сервер метрик не стартует, проверь его конфигурацию. По этой ссылке расписано, как его настроить.

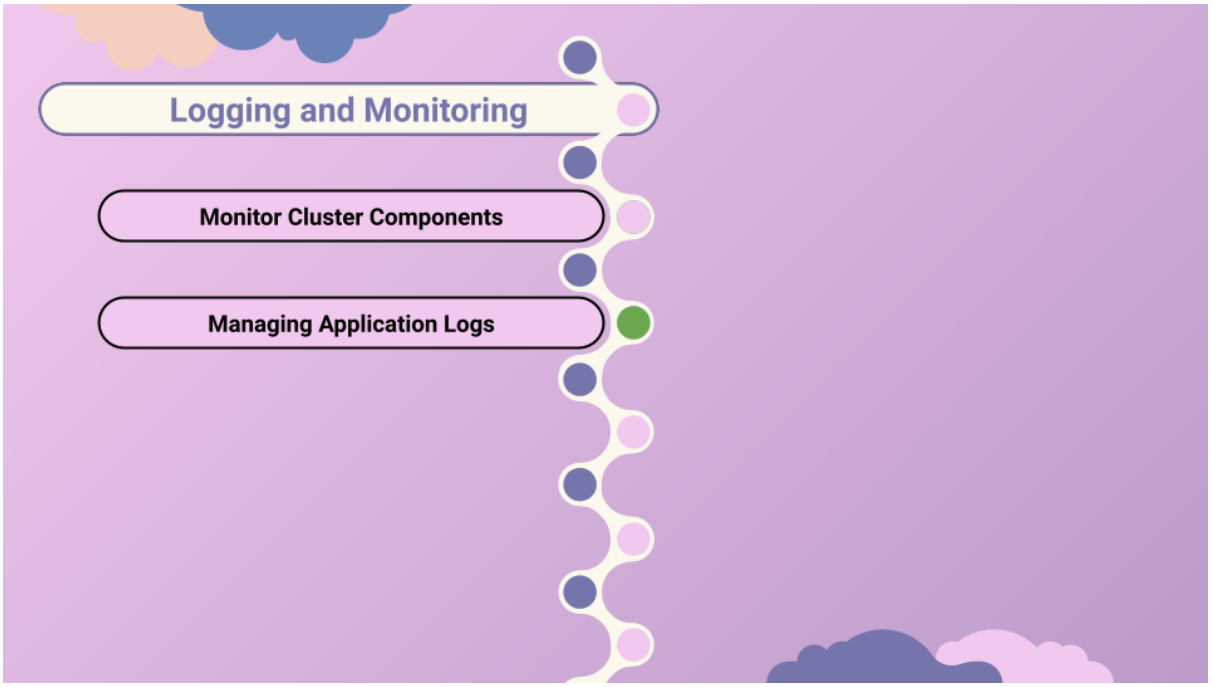
От себя скажу, что в 90% случаев это проблемы с сертификатами. Ты можешь выпустить отдельные для Metrics-Server или использовать сертификаты kube-apiserver.

Для этого отредактируй deployment в файле `components.yaml`, не забудь подключить тома с сертификатами в контейнер сервера метрик. Либо используй быстрый обходной вариант - ключ --kubelet-insecure-tls.

На этом лекция закончилась. Перейди в раздел упражнений и набей руку просматривании метрик производительности в кластере Kubernetes.

Увидимся в следующей лекции!





Привет и добро пожаловать эту лекцию.
В этой лекции мы поговорим о механизмах ведения логов в Kubernetes.

Начнем с логгинга в Docker.

Для курса я изготовил простое приложение, которое создает случайные события и отправляет на стандартный вывод, имитируя веб-сервер. Посмотри на github rotoro-cloud если тебе интересно.

Итак, мы запускам докер-контейнер под названием log-generator.

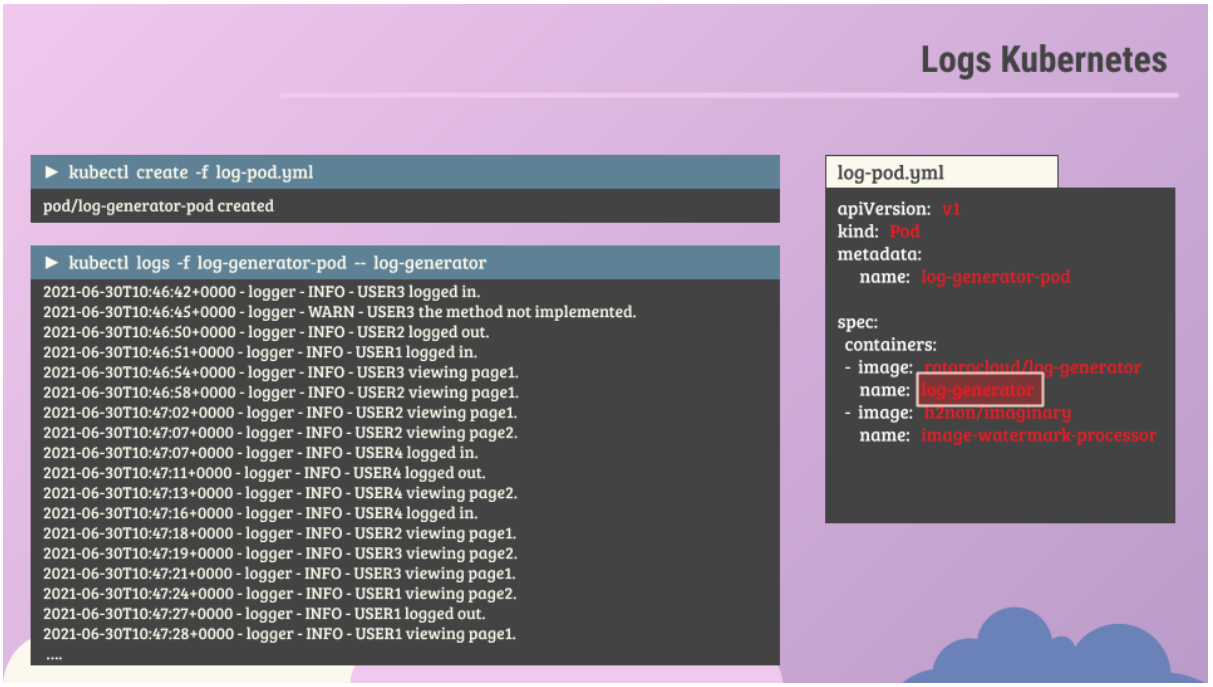
Приложение сразу начинает создавать вывод событий, создавая видимость логов веб-сервера. Оно передает их на стандартный вывод. Если ты не в курсе, то передача логов на стандартный вывод - это общепринятая практика в контейнерном мире.

Теперь, если я запущу контейнер Docker в фоновом режиме, в detached mode с параметром –d, я не увижу эти логи.

Если требуется просмотреть журнал логов, то мы используем команду docker logs, за которой идет идентификатор контейнера. Параметр –f дает возможность наблюдать за логами контейнера в реальном времени.

Теперь вернемся в Kubernetes. Мы создаем POD с тем же докер-образом, используя файл определения, как ты видишь тут.

Когда POD запущен, мы можем просмотреть его логи с помощью команды kubectl logs с именем POD.



Используя такой же параметр –f мы получим стрим логов из POD. Т.е. сможем наблюдать их в реальном времени, также, как и с командой docker в прошлый раз.

Обрати внимание, эти логи относятся к контейнеру, работающему внутри POD.

Как мы узнали ранее, в Kubernetes POD может содержать несколько контейнеров. Я изменю файл определения POD, чтобы задействовать дополнительный контейнер с именем image-watermark-processor.

Пересоздадим POD. Если сейчас мы запустим команду kubectl logs с именем POD, журнал логов какого контейнера будет отображаться?

Если у нас мульти-контейнерный POD, мы должны явно указать имя контейнера в команде.

В противном случае команда не сработает, и kubectl попросит тебя указать имя контейнера.

В этом случае я напишу команду `kubectl logs -f log-generator-pod`, а далее, чтобы указать контейнер ввожу ` -- ` и имя первого контейнера генератора логов. Теперь мы получим сообщения журнала.

Это простая функция ведения логов, реализованная в Kubernetes.

И это все, что действительно нужно знать разработчику приложения, чтобы начать работу с Kubernetes.

Но это далеко не все, что нужно знать администратору Kubernetes об этом.

Тем не менее, в рамках программы сертификации это все знания, которые требуются в домене `monitoring`.

Тема `Observability` в Kubernetes очень широкая. Подобно тому, как мы сравнивали Kubernetes с конструктором, решения мониторинга - это как покупка набора новых деталей, по объему чуть ли не равная самому конструктору.

Из-за обилия объектов в кластере и их динамичности, мониторинг в Kubernetes может поражать причудливым сплетением компонентов. Надеюсь, я смогу рассказать об этом в курсе `Prometheus - мониторинг и логирование в Kubernetes`.

Ну вот и все в этой лекции.
Перейди в раздел упражнений и попрактикуй в работе с логами.

Жду тебя на следующей лекции.