Модуль 5, урок 1

Этот урок посвящен мониторингу в целом и мониторингу микросервисных архитектур в частности. Вы узнаете, зачем, что и как мониторить, а также какие инструменты используются для мониторинга микросервисных архитектур.

Существует **золотое правило ленивых людей**: *прежде чем что-то делать, убедись, что это не пустая трата времени*.

Чтобы правильно организовать мониторинг стека технологий K8s, необходимо понимать, зачем этот мониторинг нужен, что конкретно собираетесь мониторить. А затем, исходя из этих знаний, найти правильные инструменты и способы организации мониторинга.

Kubernetes и микросервисы

Перед тем как мы перейдем непосредственно к мониторингу, поговорим про сам Kubernetes и микросервисы.

Зародившийся когда-то модный тренд уже перестал быть просто модным. Микросервисная архитектура с нами надолго. И это факт! Принято считать, что она интересна только разработчикам. Однако это не совсем верно.

Многие поставщики программных решений начали представлять свои продукты в виде микросервисов. И вдруг повсеместно новые версии Enterprise-решений стали переезжать на микросервисные рельсы.

А ИТ-администраторы, которые и думать не хотели о Kunernetes, вдруг стали обязаны предоставлять и обслуживать инфраструктуру под микросервисные приложения. Представьте их удивление и возросший объем работы!

Микросервисы в целом — это результат требований к скорости процесса разработки и внедрения.

Мир не стоит на месте, цифровые продукты развиваются сумасшедшими темпами. Выигрывает гонку тот, кто быстрее выпустил новый продукт на рынок или предоставил новую фичу раньше конкурентов. Это постоянно ускоряющийся процесс, который требует хорошо отлаженной обратной связи. Для работы этого процесса — в сути самой идеологии DevOps — требуется постоянно улучшать обратную связь и использовать ее для оптимизации приложения или сервиса. Львиную долю этой обратной связи как раз обеспечивает мониторинг.

Но как правильно этот мониторинг организовать? Ведь микросервисная архитектура — это большой и сложный слоеный пирог из целого ряда различных технологий.

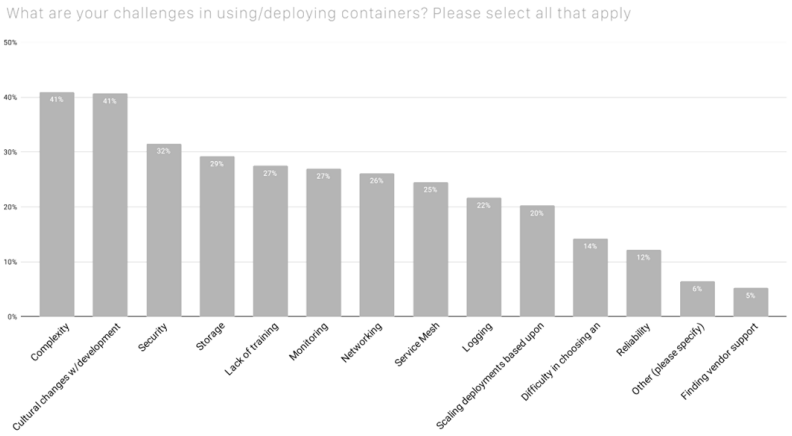
Мониторинг ради мониторинга не работает. И не заработает никогда. В первую очередь нужно понимать, зачем он нужен.

Наиболее распространенные задачи мониторинга:

- поиск неисправностей и узких мест;
- отслеживание трендов утилизации ресурсов;
- отслеживание эффективности работы приложений и сервисов;
- сбор и анализ метрик работы самого приложения.

Казалось бы, раз мы знаем, зачем нужен мониторинг и что конкретно мы хотим мониторить, дело остается за малым — за реализацией. Однако тут сложность задачи как раз возрастает. Cloud Native Compute Foundation ежегодно проводит опрос, какие сложности возникают при использовании микросервисов. Мониторинг и логирование — одни из наиболее частых ответов.

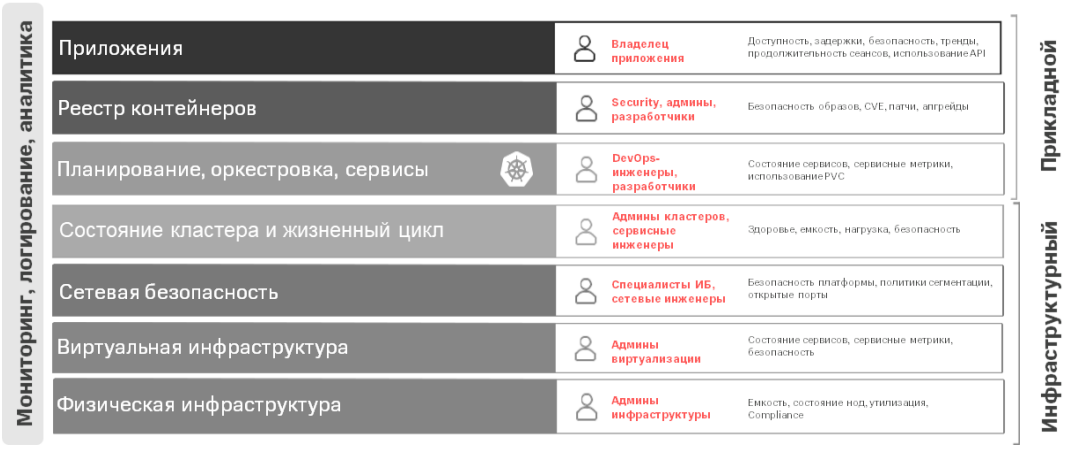
***Данные из опроса CNCF**



Почему DevOps-инженеры и администраторы Kubernetes так отвечают?
Сложности возникают из требований к самой платформе. Чтобы правильно мониторить платформу, она должна, в первую очередь, предоставлять такую возможность. Это та самая наблюдаемость — какие данные нам нужно вытащить из системы, которую хотим мониторить. Дальше нужен правильный инструмент для сбора, агрегирования, обработки и отображения данных. Это очень важный момент. Нельзя забывать про анализ — автоматическую или в крайнем случае ручную обработку собранных данных для придания им ценности и дальнейшего принятия каких-либо мер.

Но это ведь еще и не вся картина целиком. Живая инфраструктура — это сложный многоуровневый организм. Обслуживают его разные команды с разными задачами. И таких команд может быть множество, причем их интересы и зоны ответственности чаще всего размыты и пересекаются между собой.

Кто мониторит? Что мониторит?



- На прикладном уровне владельцы приложений несут ответственность за работу своих приложений перед бизнесом и конечным пользователем.
- На инфраструктурном уровне владельцы платформ и администраторы несут ответственность за предоставление платформы своим потребителям. Их главная забота — поддержание доступности, производительности и емкости платформы для обеспечения потребностей владельцев приложений. Им необходимо понимать, насколько эффективно работает платформа, есть ли какие-то ошибки или узкие места.

- Нельзя забывать про специалистов информационной безопасности. У них тоже есть ряд требований к самой платформе, инструментам и способам реализации.

Попробуем посмотреть на ситуацию глазами основных участников — администратора инфраструктуры и владельца приложений.

- Нижний слой этого стека — физические серверы, устройства хранения данных и сети. Это может быть несколько внезапно, но serverless тоже работает на серверах.
- Чуть выше — программный слой абстракции. Те самые гипервизоры, которые позволяют запускать несколько параллельных нагрузок на одном физическом сервере.
- Поверх всего этого работает ряд виртуальных машин, которые могут нести (или не нести) в себе Kubernetes или интегрироваться с ним.
- Работают в них, конечно же, сами приложения, которые для многих компаний представляют собой смесь устаревших приложений с традиционной архитектурой и современных микросервисов.
- И, наконец, эти приложения представляют собой некий сервис для бизнеса.

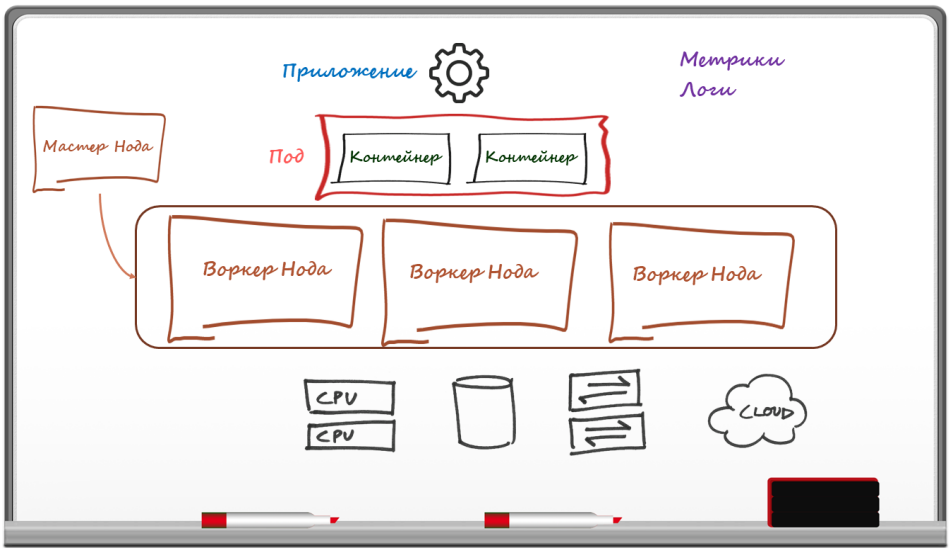
За нижнюю часть стека традиционно отвечают **операторы и администраторы** (те, кто предоставляет инфраструктуру для использования потребителями). В верхней части стека находятся как раз потребители инфраструктуры — владельцы приложений. На стороне инфраструктуры необходим **мониторинг** для поиска и устранения неполадок, то есть полная видимость всех инфраструктурных элементов. Именно это сегодня предлагают vRealize Operations и Log Insight. Они же и обеспечивают необходимый функционал по расчету емкости, стоимости и производительности.

Владельцы приложений ближе к бизнесу и отвечают непосредственно за приложения. Их совсем не интересует инфраструктура и все, что с ней происходит. Главное — чтобы приложение работало и работало хорошо. Для отслеживания метрик приложения они используют такие open-source-инструменты, как, например, Prometheus, который обеспечивает очень точный сбор и анализ всех метрик окружения на прикладном уровне.

Это исторически сложившиеся рамки, но они уже **не отвечают требованиям современности**. Эти рамки стираются, и их больше нельзя рассматривать как нечто отдельное и взаимоисключающее.

По сути, те, кто отвечает за инфраструктуру, все равно должны знать и о приложениях, ведь инфраструктура в вакууме бессмысленна: нужно знать, как это влияет на бизнес. Точно так же потребитель связан с приложением, но он по-прежнему должен быть осведомлен об инфраструктуре.

Если переложить эту картину на микросервисную архитектуру, становится понятна сложность поставленной задачи.

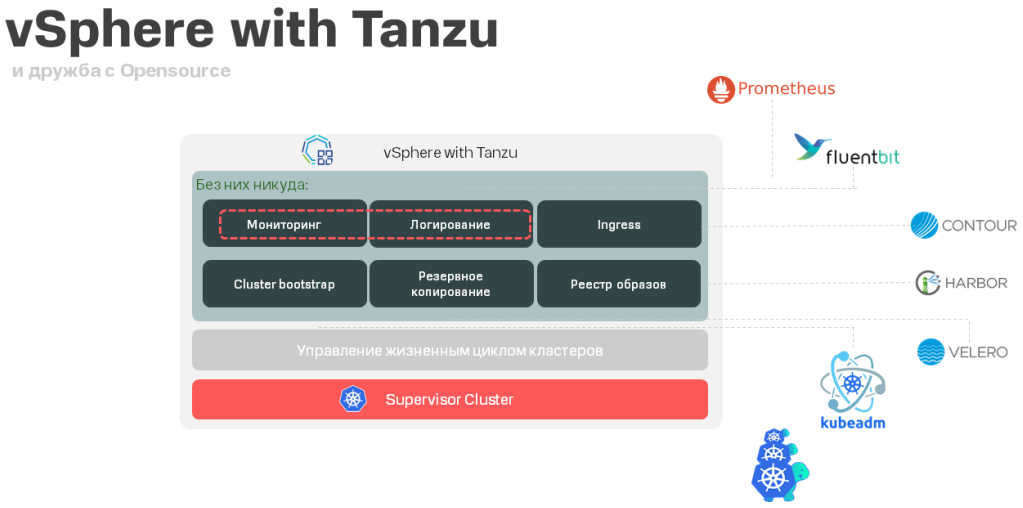


На верхнем уровне микросервисная архитектура выглядит так:

- Существует некое приложение, выполняющее определенный функционал. Оно может обслуживать пользователей или другие приложения.
- В микросервисной архитектуре приложение работает внутри контейнера.
- Оно может состоять сразу из нескольких контейнеров, объединенных в некий под. Под — это минимальная единица, доступная для запуска в мире K8s. Под может состоять из одного, но чаще состоит из нескольких контейнеров, объединенных логически и физически, с единой сетью и ресурсами хранения данных (если они требуются).
- Поды работают в пространствах имен, выделенных на кластерах K8s. Кластер состоит из одной или нескольких master-нод и одной или нескольких worker-нод. Это могут быть как физические bare-metal-установки, так и виртуальные машины.
- Этот кластер работает на каких-то вычислительных ресурсах, которые очень часто представляют собой несколько уровней абстракции: железо, гипервизор, инфраструктурный провайдер.
- Каждый из этих компонентов и слоев генерирует две интересные для мониторинга сущности — метрики и логи.

Для сбора метрик и логов существует масса инструментов. Мы сконцентрируемся на двух:

- Prometheus — для метрик;
- Fluent Bit — для логов.



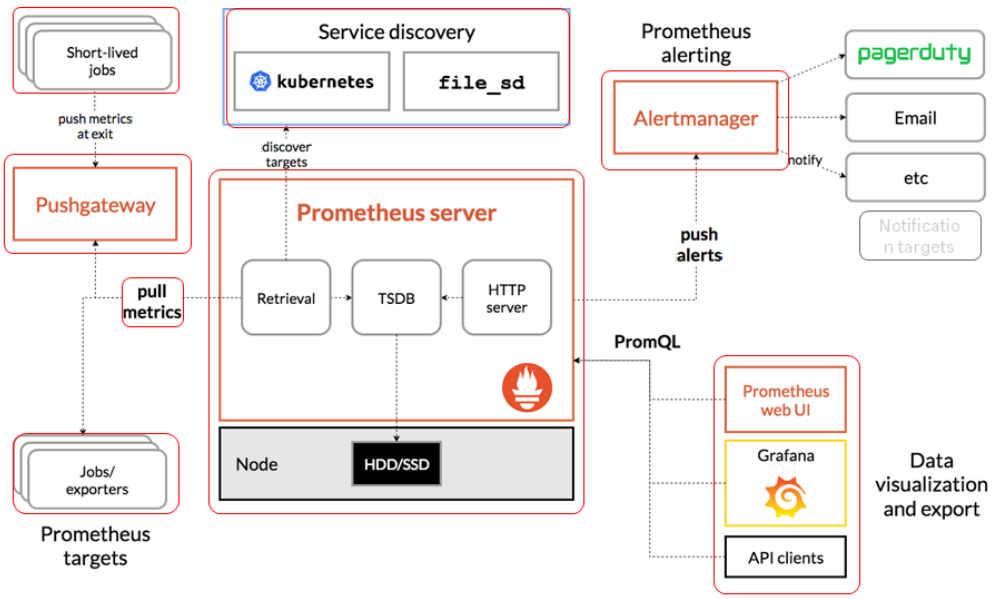
Это инструменты с открытым исходным кодом, поставляемые и поддерживаемые в рамках vSphere with Tanzu.

Анатомия метрики

Метрики — это некоторые значения переменных во времени. Их формат обычно короткий и включает имя, значение, метку времени и источник. При желании он может содержать другие точечные теги, которые помогут организовать данные удобным способом. Примерами таких точечных тегов являются идентификаторы порта, хоста или контейнера. Их преимущество в том, что их можно легко хранить и обрабатывать. Собирать их и обрабатывать намного быстрее и проще, чем логи.

Логи — это длинный кусок текста, в котором может быть множество мусорной информации. Но именно логи чаще всего дают максимальную информативность и данные, которые можно использовать для глубокого анализа.

Prometheus

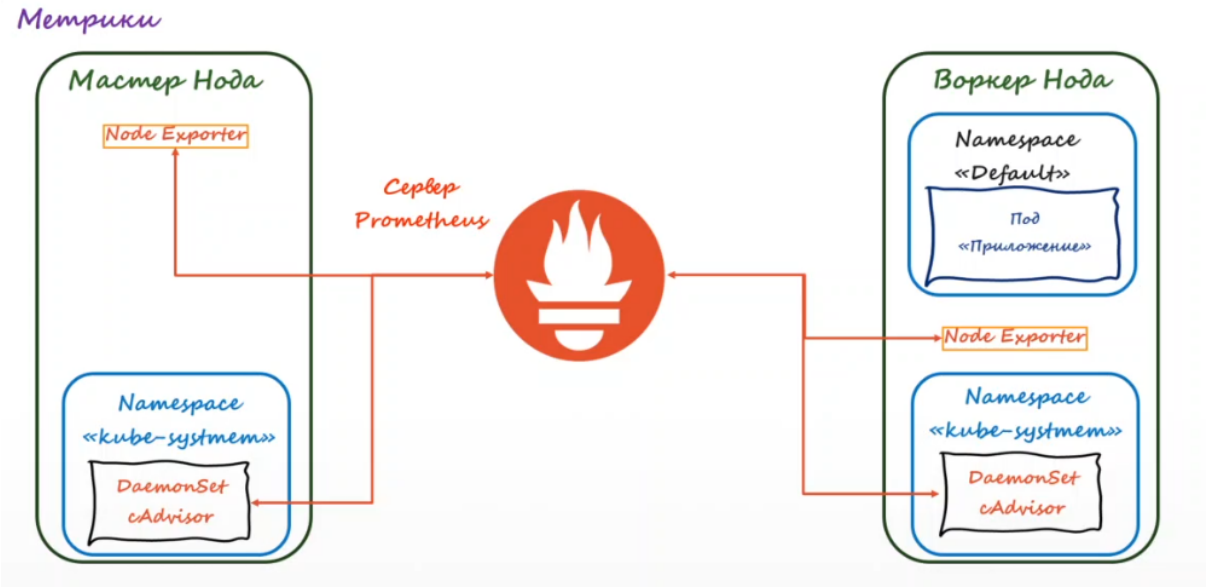


Для сбора и обработки метрик есть замечательный инструмент — Prometheus. Это проект с богатой историей. Он появился на свет в 2012 году и с тех пор успел завоевать любовь open-source-сообщества, особенно после того, как присоединился к Cloud Native Compute Foundation. Кстати говоря, вторым после самого Kubernetes.

Это, наверное, самый популярный инструмент для мониторинга K8s-инфраструктур. Разберемся, как он работает:

1. Prometheus — это не модуль и не агент, а полноценный многомодульный приклад серверного типа.
2. Метрики он собирает сам с определенных целей, которые прописаны вручную или получены с помощью service discovery. Это избавляет от потенциальных проблем с невозможностью какой-то цели экспортировать метрики самостоятельно.
3. Данные собираются напрямую, если цель сбора метрик поддерживает такой сценарий, или с помощью специализированных экспортеров, если цель не поддерживает интеграцию с Kubernetes.
4. Бывают сценарии, когда требуется снимать метрики с короткоживущих процессов. Задачу получения таких метрик тоже можно решить. Для этого используется отдельный компонент Push Gateway. Это своего рода прокси-компонент, который получает данные с короткоживущих процессов и отдает их Prometheus.
5. Как и любой правильный мониторинг, Prometheus может отправлять оповещения во внешние системы с помощью дополнительного компонента Alertmanager. Это могут быть как обычные email-уведомления, так и какие-то корпоративные чат-системы.
6. Prometheus может отдавать данные на анализ и отрисовку во внешние системы.

Сервер Prometheus собирает и хранит метрики, добавляя временную метку, в которую метрика была записана. Собирает он их с разных элементов и разными способами.



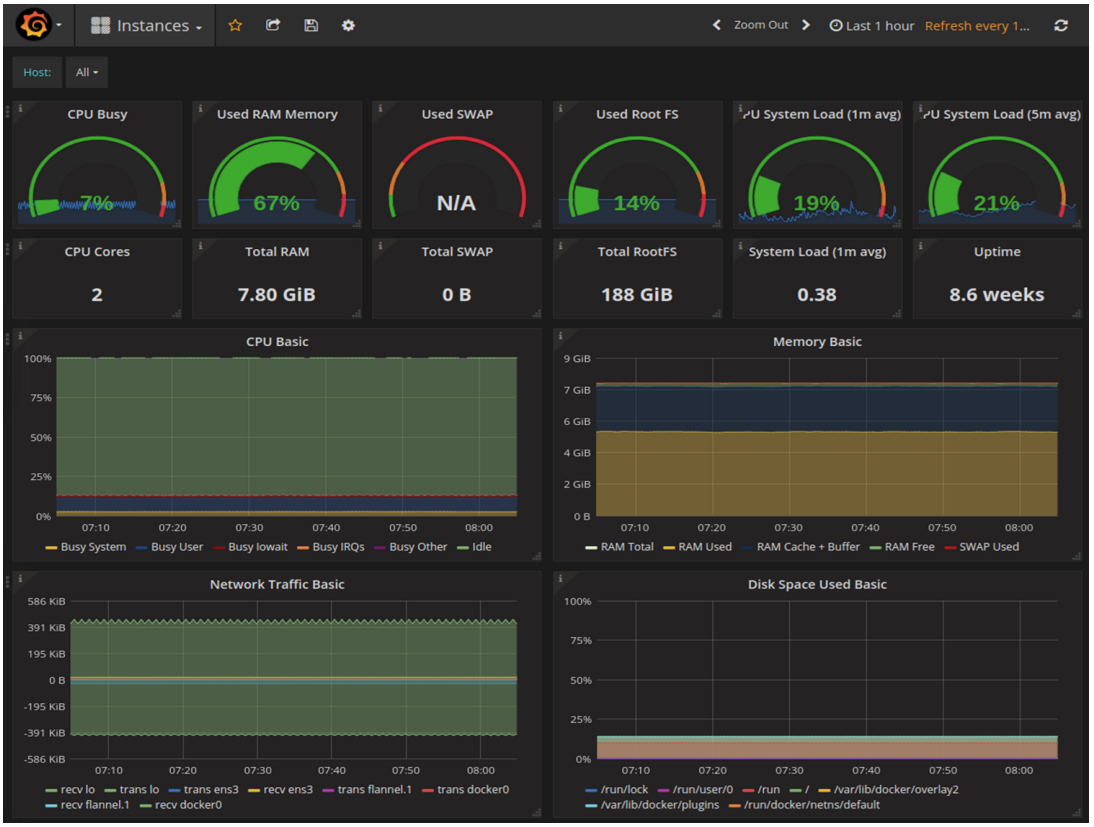
1. K8s-ноды — master и worker. Node exporter собирает данные, доступные на уровне корневой ОС: загрузку, задержки, процессы, статистику работы и ошибок нижележащих аппаратных компонентов — то есть все, что может отдать базовая ОС.

2. Статистика работы контейнеров в виде потребления ресурсов — процессорной емкости, памяти, сети. Это может собирать компонент cAdvisor, который чаще всего запускают как DaemonSet в кластере K8s. Он доступен прямо из коробки. DaemonSet — это сущность, которая запускается сразу на всех нодах кластера. Де-факто — тоже контейнерное приложение, но отвечающее за какие-то системные ресурсы или сервисы.

3. Приложения. Prometheus может собирать данные с приложений, в которые уже встроена его нативная поддержка. Те приложения, где ее нет, тоже можно опрашивать с помощью компонента под названием «экспортер». Это side-car-контейнер, который запускается в том же поде, что и само приложение. Экспортеров на текущий момент есть огромное множество — фактически под все популярные open-source-приложения.

4. Все эти данные можно отрисовать с помощью Grafana и интегрировать Prometheus в vRealize Operations для сбора и обработки этих метрик там.

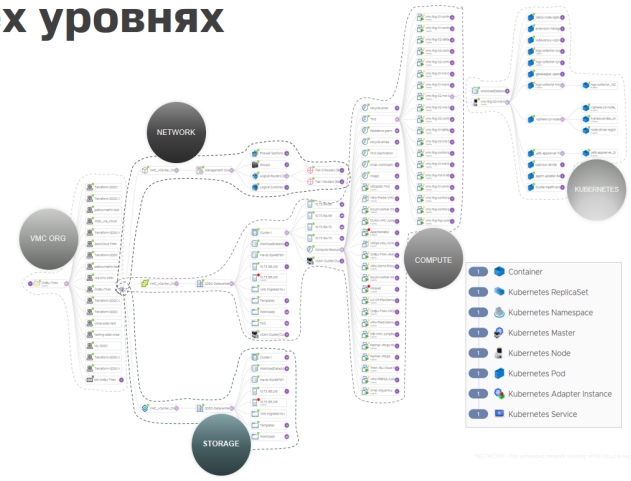
Зачем данные отправлять куда-то еще?
Следует понимать, как Prometheus видит эти данные. Это сырой набор различных метрик, с которыми невозможно работать руками. Сам Prometheus умеет рисовать базовые графики, но обычно этого недостаточно. Их необходимо привести в визуально понятный формат. Тут как раз и пригодится Grafana, которая может отрисовать их так, как вам удобно. Это прекрасно работает для DevOps-команд и владельцев приложений, которым нужен конкретный срез данных по их зоне ответственности.



Но и про остальную инфраструктуру забывать нельзя. Ведь если что-то пойдет не так на других слоях этого большого слоеного пирога из различных технологий, это повлияет на работу приложений. И тут на сцену выходит vRealize Operations, который позволяет получить детальную картину взаимоотношений между всеми уровнями вложенности.

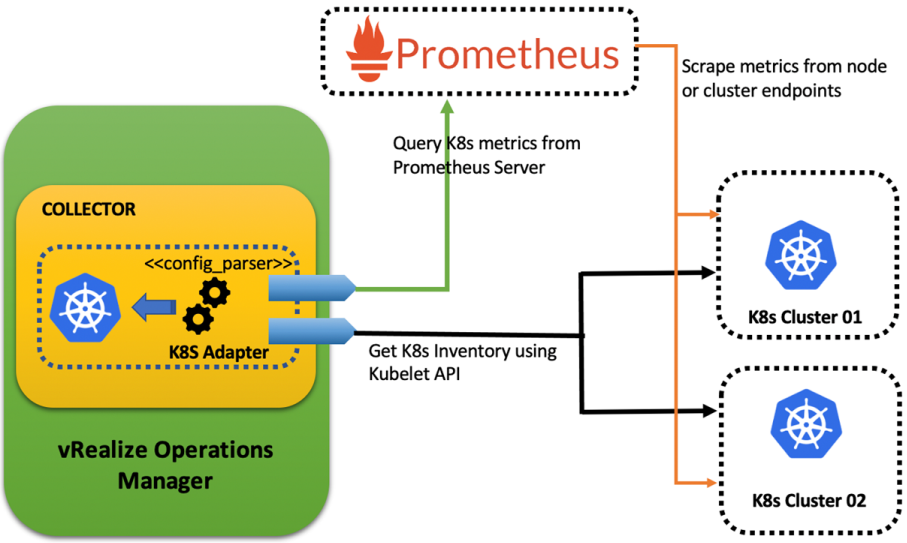
Kubernetes to Infrastructure – видимость на всех уровнях

vRealize Operations



Пример. Во главе vCenter — сервер с кластером виртуализации vSphere, на котором расположен пул ресурсов, где работает кластер K8s. Мы можем раскрыть детали самого K8s, увидеть узел, поды и контейнеры. Это дает нам возможность детально и точно искать проблемы, находить корреляции между различными метриками разных уровней вложенности. Если вдруг у вас вырастают задержки на уровне POD или BM и это связано с задержками на хосту — эти корреляции можно будет найти. Вы точно сможете определить, влияют ли проблемы в инфраструктуре на работу приложений.

Давайте разберемся, как это работает:



Есть замечательный Management Pack для K8s, который можно импортировать и установить в vRealize Operations. Он добавляет новый адаптер, который, подключаясь к Kube API, позволяет собирать статистику и объекты, существующие в рамках кластера Kubernetes. В связке с Prometheus в vRealize Operations собирает метрики по этим объектам, сопоставляя их с текущей инфраструктурой.