**МОНИТОРИНГ КЛАСТЕРА АНАЛИЗА БОЛЬШИХ ДАННЫХ APACHE SPARK НА ОСНОВЕ KUBERNETES**

*Загребаев Дмитрий Кириллович*

*Студент факультета «Информационные системы и телекоммуникации»*

*ФГБОУ ВО «Московский государственный технический университет*

*имени Н.Э. Баумана*

*(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)*

Kubernetes является большим проектом с открытым исходным кодом, а также экосистемой с гигантским кодом и громадной функциональностью. Kubernetes был сделан Google, однако присоединился к **CNCF** (**Cloud Native Computing Foundation**, Фонд присущих облакам вычислений) и стал очевидным лидером в данной области приложений на базе контейнеров. Если выразить одним предложением, это платформа для организации развёртывания, масштабирования и управления контейнерными приложениями.

# **Kubernetes: система управления контейнерами**

Kubernetes является проектом с открытым исходным кодом, предназначенным для управления кластером контейнеров Linux как единой системой. Kubernetes управляет и запускает контейнеры Docker на большом количестве хостов, а также обеспечивает совместное размещение и репликацию большого количества контейнеров. Проект был начат Google и теперь поддерживается многими компаниями, среди которых Microsoft, RedHat, IBM и Docker.

Компания Google пользуется контейнерной технологией уже более десяти лет. Она начинала с запуска более 2 млрд контейнеров в течение одной недели. С помощью проекта Kubernetes компания делится своим опытом создания открытой платформы, предназначенной для масштабируемого запуска контейнеров.

Проект преследует две цели. Если вы пользуетесь контейнерами Docker, возникает следующий вопрос о том, как масштабировать и запускать контейнеры сразу на большом количестве хостов Docker, а также как выполнять их балансировку. В проекте предлагается высокоуровневый API, определяющее логическое группирование контейнеров, позволяющее определять пулы контейнеров, балансировать нагрузку, а также задавать их размещение.

Основные разработчики первых версий внутренней системы Google — программисты Джо Беда (*Joe Beda*), Брендан Бёрнс (*Brendan Burns*) и Крэйг Маклаки (*Craig McLuckie*), в дальнейшем к проекту присоединились их коллеги Брайан Грант (*Brian Grant*) и Тим Хокин (*Tim Hockin*). Основной язык программирования системы — Go. На разработку и внутреннюю идеологию Kubernetes серьёзно повлиял другой продукт Google, оставшийся внутренней разработкой — система управления кластерами Google Borg, над которым ранее работал ряд ключевых разработчиков Kubernetes.

Оригинальное наименование проекта — *Project Seven* (отсылка к героине сериала Star Trek, возвращённой в индивидуальное и дружественное к людям состояние из статуса члена нечеловеческого роевого кибернетического разума Коллектива Боргов); семь ручек на штурвале логотипа проекта — аллюзия на этот художественный образ.

В середине 2014 года опубликованы исходные коды проекта. 21 июля 2015 года выпущена версия 1.0; после чего Google в партнёрстве с Linux Foundation организован специальный фонд Cloud Native Computing Foundation (CNCF), которому корпорация передала Kubernetes в качестве начального технологического вклада.

Как и многие другие сложные продукты, Kubernetes в рамках своей экосистемы вводит ряд специфических понятий и концепций. Ниже перечислены основные концепции рассматриваемого кластера.

***Узел (node)*** — это отдельная физическая или виртуальная машина, на которой развёрнуты и выполняются контейнеры приложений. Каждый узел в кластере содержит сервисы для запуска приложений в контейнерах (например, Docker), а также компоненты, предназначенные для централизованного управления узлом.

***Под (pod, с англ. — «стручок, кокон»)*** — базовая единица для управления и запуска приложений, один или несколько контейнеров, которым гарантирован запуск на одном узле, обеспечивается разделение ресурсов и предоставляется уникальный в пределах кластера IP-адрес. Последнее позволяет приложениям, развёрнутым на поде, использовать фиксированные и предопределённые номера портов без риска конфликта. Поды могут напрямую управляться с использованием API Kubernetes или управление ими может быть передано контроллеру.

***Том (volume)*** — общий ресурс хранения для совместного использования из контейнеров, развёрнутых в пределах одного пода.

Все объекты управления (узлы, поды, контейнеры) в Kubernetes помечаются ***метками (label), селекторы меток (label selector)*** — это запросы, которые позволяют получить ссылку на объекты, соответствующие какой-то из меток; метки и селекторы — это главный механизм Kubernetes, который позволяет выбрать, какой из объектов следует использовать для запрашиваемой операции.

***Сервисом*** в Kubernetes называют совокупность логически связанных наборов подов и политик доступа к ним. Например, сервис может соответствовать одному из уровней программного обеспечения, разработанного в соответствии с принципами многоуровневой архитектуры программного обеспечения. Набор подов, соответствующий сервису, получается в результате выполнения селектора соответствующей метки.

Kubernetes обеспечивает функции обнаружения сервисов и маршрутизации по запросу, в частности, система умеет переназначать необходимые для обращения к сервису IP-адрес и доменное имя сервиса различным подам, входящим в его состав. При этом обеспечивается балансировка нагрузки между подами, чьи метки соответствуют сервису в стиле Round robin DNS, а также корректная работа в том случае, если один из узлов кластера вышел из строя и размещённые на нём поды автоматически переместились на другой. По умолчанию сервис доступен внутри управляемого Kubernetes кластера, например, поды бэкенда группируются для обеспечения балансировки нагрузки и в таком виде предоставляются фронтенду, но он может быть настроен и для того, чтобы предоставлять доступ к входящим в его состав подам извне, как к единому фронтенду.

***Контроллер (controller)*** — это процесс, который управляет состоянием кластера, пытаясь привести его от фактического к желаемому; он делает это, оперируя набором подов, который определяется с помощью селекторов меток, являющихся частью определения контроллера. Выполнение контроллеров обеспечивается компонентом Kubernetes Controller Manager. Один из типов контроллеров, самый известный — это контроллер репликации (*Replication Controller*), который обеспечивает масштабирование, запустив указанное количество копий пода в кластере. Он также обеспечивает запуск новых экземпляров пода, в том случае если узел, на который работает управляемый этим контроллером под, выходит из строя. Другие контроллеры, входящие в основную систему Kubernetes, включают в себя «DaemonSet Controller», который обеспечивает запуск пода на каждой машине (или подмножеством машин) и «Job Controller» для запуска подов, которые выполняются до завершения, например, как часть пакетного задания.

***Операторы (operators)*** — специализированный вид программного обеспечения Kubernetes, предназначенный для включения в кластер сервисов, сохраняющих своё состояние между выполнениями (*stateful*, таких как СУБД, системы мониторинга или кэширования. Назначение операторов — предоставить возможность управления stateful-приложениями в кластере Kubernetes прозрачным для способа и скрыть подробности их настроек от основного процесса управления кластером Kubernetes.

Система реализует архитектуру «ведущий — ведомый»: выделяется подсистема управления кластером, а часть компонентов управляют индивидуальным, ведомыми узлами (называемых собственно узлами Kubernetes). Приведённая схема показывает желаемое, в конечном итоге, состояние, хотя все ещё ведётся работа над некоторыми вещами, например, как сделать так, чтобы kubelet (все компоненты, на самом деле) самостоятельно запускался в контейнере, что сделает планировщик на 100% подключаемым.

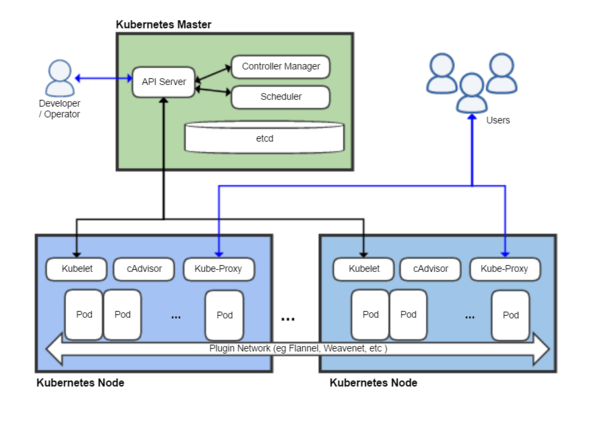


Рис. 1. Схема архитектуры Kubernetes

Подсистема обеспечивает управление распределением нагрузки и коммуникациями внутри кластера и состоит из следующих приложений (Рис. 1). Компоненты подсистемы управления могут выполняться на одном или на нескольких параллельно работающих мастер-узлах, совместно обеспечивающих режим высокой доступности.

***Etcd*** — компонент подсистемы управления, отвечающий за согласованное хранение конфигурационных данных кластера, в некотором смысле — распределённый эквивалент каталога /etc Unix-систем. Реализован как легковесная распределённая NoSQL-СУБД класса «ключ — значение»; создан в рамках проекта CoreOS.

***Сервер API***— ключевой компонент подсистемы управления, предоставляющий API в стиле REST (с использованием коммуникации в формате JSON поверх HTTP-транспорта), и используемый для организации внешнего и внутреннего доступа к функциям Kubernetes. Сервер API обновляет состояние объектов, хранящееся в etcd, позволяя своим клиентам управлять распределением контейнеров и нагрузки между нодами управляемой системы.

***Планировщик (scheduler)***— компонент подсистемы управления, который выбирает, на каком узле должен выполняться конкретный под, опираясь на критерии доступности ресурсов. Планировщик отслеживает использование ресурсов на каждом из узлов, обеспечивая распределение нагрузки так, чтобы она не превышала доступный объём ресурсов. Для этой цели планировщик должен обладать информацией о доступных на каждом из узлов ресурсах, требованиях к ним со стороны управляемых подов, а также различных дополнительных пользовательских ограничениях и политиках, таких как QoS, требования аффинитета и антиаффинитета (*affinity — anti-affinity* — связки или развязки объектов управления друг с другом), локализации данных. Иными словами, роль планировщика — находить и предоставлять ресурсы в зависимости от запросов, возникающих в связи с загрузкой.

***Менеджер контроллеров (controller manager)***— процесс, выполняющий основные контроллеры Kubernetes, такие как DaemonSet Controller и Replication Controller run in. Контроллеры взаимодействуют с сервером API Kubernetes, создавая, обновляя и удаляя управляемые ими ресурсы (поды, точки входа в сервисы, и другие).

***Kubectl*** — интерфейс командной строки, наряду с API обеспечивающий управление ресурсами, подконтрольными Kubernetes.

Процедура работы Kubernetes состоит в том, что ресурсы нод Kubernetes динамически распределяются между выполняемыми на них подами. Каждый узел в кластере Kubernetes содержит ряд типовых компонентов.

Сервис для запуска контейнеров обеспечивает функции выполнения контейнеров соответствующего вида (в зависимости от типа используемого контейнерного движка). С точки зрения программной среды Kubernetes, контейнеры инкапсулируются в подах, при этом сами контейнеры являются наиболее низкоуровневыми программными компонентами, с которыми взаимодействует программное обеспечение Kubernetes. Они, в свою очередь, содержат выполняемые приложения, библиотеки и иные необходимые для работы этих приложений ресурсы. Для внешнего мира контейнеры доступны через назначаемый каждому из контейнеров IP-адрес.

***Kubelet*** отвечает за статус выполнения подов на узле — отслеживает, корректно ли выполняется каждый из контейнеров, находясь в рабочем состоянии. Kubelet обеспечивает запуск, останов и управление контейнерами приложений, организованными в поды. Функционально Kubelet можно рассматривать как аналог supervisord. Если обнаруживается, что какой-то из подов находится в неверном состоянии, компонент пытается осуществить его повторное развёртывание и перезапуск на узле. Статус самого узла отправляется на подсистеме управления каждые несколько секунд в форме диагностических сообщений (*heartbeat message*). Если мастер-узел, исходя из содержания этих сообщений или их отсутствия, обнаруживает, что конкретный узел не работает должным образом, процесс подсистемы управления Replication Controller пытается перезапустить необходимые поды на другом узле, находящемся в рабочем состоянии.

***Kube-proxy***— компонент, являющийся комбинацией сетевого прокси-сервера и балансировщика нагрузки. Реализованные в нём операции сетевого уровня используют абстракцию сервиса. Он отвечает за маршрутизацию входящего трафика на конкретные контейнеры, работающие в пределах пода, расположенного на узле. Маршрутизация обеспечивается на основе IP-адреса и порта входящего запроса.

***cAdvisor***— агент системы внутреннего мониторинга Kubernetes, собирающий метрики производительности и информацию об использовании контейнерами, работающими в пределах ноды, таких ресурсов как время работы центрального процессора, оперативной памяти, нагрузку на файловую и сетевую системы.

# **Apache Spark: кластерная вычислительная платформа**

**Apache Spark** (от англ. *spark* — искра, вспышка) — фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с дисковым хранилищем, Spark использует специализированные примитивы для рекуррентной обработки в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Проект предоставляет программные интерфейсы для языков Java, Scala, Python. Изначально написан на Scala, впоследствии добавлена существенная часть кода на Java для предоставления возможности написания программ непосредственно на Java. Состоит из ядра и нескольких расширений, таких как Spark SQL (позволяет выполнять SQL-запросы над данными), Spark Streaming (надстройка для обработки потоковых данных), Spark MLlib (набор библиотек машинного обучения), GraphX (предназначено для распределённой обработки графов). Может работать как в среде кластера Hadoop под управлением YARN, так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

Резюмируя, можно назвать причины, по которым данный фреймворк заслужил такое внимание:

* Во-первых, **скорость работы** на всех уровнях фреймворка – от запросов SQL до вычислений на графах и машинного обучения. Цифра 100х зацепила многих, и в особенности представителей бизнеса, для которых время – это деньги в прямом смысле слова.
* Во-вторых, **многофункциональность.** Действительно, рассмотрев подробнее архитектуру Spark, мы с вами можем прийти к выводу, что столь богатые возможности одного-единственного приложения не могут остаться без внимания.
* В-третьих, Spark является **модификацией MapReduce** в том смысле, что с внедрением нового фреймворка пользователи не рискуют такими существенными характеристиками предшественника как неограниченная горизонтальная масштабируемость и способность восстанавливаться даже после серьезных ошибок системы, а также полная интеграция с экосистемой Hadoop и, как следствие, возможность работы с существующими компьютерами и данными.

Ключевой автор — румынско-канадский учёный в области информатики Матей Захария (англ. *Matei Zaharia*), начал работу над проектом в 2009 году, будучи аспирантом Университета Калифорнии в Беркли. В 2010 году проект опубликован под лицензией BSD, в 2013 году передан фонду Apache и переведён на лицензию Apache 2.0, в 2014 году принят в число проектов верхнего уровня Apache.

Несмотря на сходство с Hadoop, Spark представляет собой новую кластерную вычислительную среду, обладающую полезными особенностями. Во-первых, Spark предназначен для решения в вычислительном кластере задач определенного типа. А именно, таких, в которых рабочий набор данных многократно используется в параллельных операциях (например, в алгоритмах машинного обучения). Для оптимизации задач этого типа в Spark вводится понятие кластерных вычислений в памяти, когда наборы данных могут временно храниться в оперативной памяти для уменьшения времени доступа.

Кроме того, в Spark вводится понятие *устойчивого распространенного набора данных* (resilient distributed datasets - RDD). RDD — это коллекция неизменяемых объектов, распределенных по множеству узлов. Эти коллекции устойчивы, потому что в случае потери части набора данных они могут восстанавливаться. Процесс восстановления части набора данных опирается на механизм отказоустойчивости, поддерживающий *родословную* (или информацию, которая позволяет восстанавливать часть набора данных с помощью процесса, в результате которого эти данные были получены). RDD представляет собой объект Scala и может создаваться из файла; в виде параллельного среза (распространенного по узлам); как преобразование другой RDD; и, наконец, путем изменения персистенции, существующей RDD, такой как запрос на кэширование в памяти.

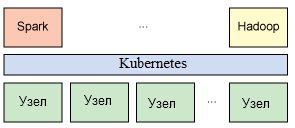


Рис. 2. Для распределения и изоляции ресурсов Spark опирается на менеджера кластера Kubernetes

В Spark приложения называются *драйверами,* и эти драйверы осуществляют операции, выполняемые на одном узле или параллельно на наборе узлов. Как и Hadoop, Spark поддерживает одноузловые и много узловые кластеры. При много узловой работе Spark опирается на менеджера кластера Kubernetes. Kubernetes обеспечивает эффективную платформу распределения ресурсов и изоляции распределенных приложений (Рис. 2). Такой подход позволяет Spark сосуществовать с Hadoop в общем пуле узлов.

# **Мониторинг кластера на основе Kubernetes**

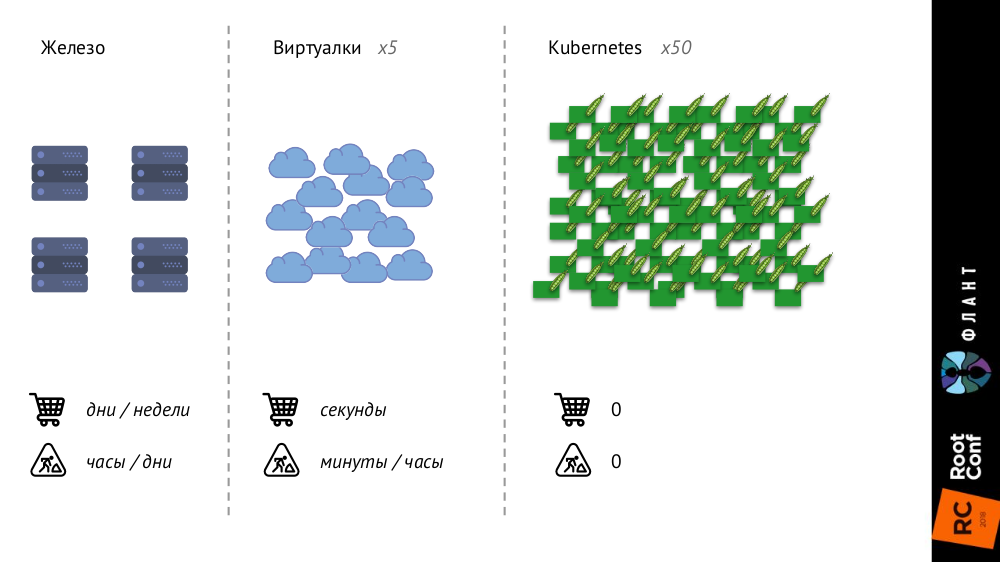
Мониторинг состоит из трех основных аспектов:

1. В первую очередь это система, позволяющая упреждать аварии, уведомлять об авариях (если их не удалось предупредить) и проводить быструю диагностику проблем.
2. Что для этого необходимо? Точные данные, полезные графики (смотришь на них и понимаешь, где проблема), актуальные алерты (приходят в нужное время и содержат понятную информацию).
3. И чтобы всё это заработало, нужна собственно система мониторинга.

1.Больше и быстрее

С Kubernetes многое меняется, потому что инфраструктура становится больше и быстрее. Если раньше, с обычными физическими серверами, их количество было достаточно ограниченным, а процесс добавления — очень долгим (занимал дни или недели), то с виртуальными машинами количество «серверов» значительно выросло, а время их введения в бой сократилось до секунд.

С Kubernetes же количество «серверов» выросло ещё на порядок (Рис. 3), их добавление полностью автоматизировано (управление конфигурациями необходимо, т.к. без описания новый pod просто не может быть создан), вся инфраструктура стала очень динамичной (например, при каждом деплое pod’ы удаляются и создаются снова).

  
Рис. 3. Эволюция развёртывания «серверов»

Мы в принципе перестаём смотреть на отдельные pod’ы или контейнеры — теперь нас интересуют только группы объектов.

Service Discovery становится строго обязательным, потому что «скорости» уже таковы, что мы в принципе не можем заводить/удалять вручную новые сущности, как это было раньше, когда новые серверы покупались.

Объём данных значительно растёт. Если раньше метрики собирались с серверов или виртуальных машин, то теперь с pod’ов, количество которых сильно больше.

Самое интересное изменение я назвал «текучкой метаданных» и расскажу о нём подробнее.

Начну с такого сравнения:

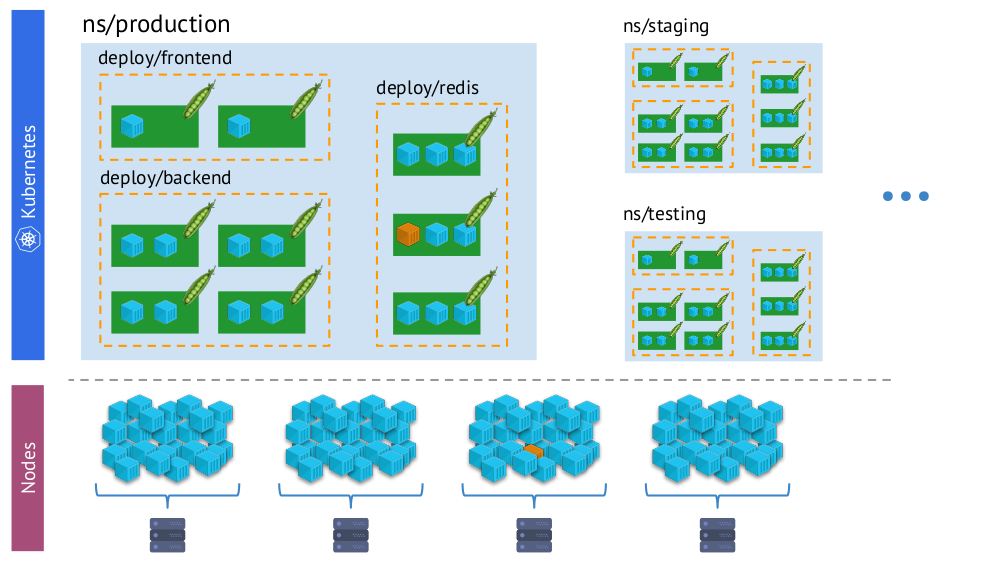
* Когда вы отдаёте ребёнка в детский садик, ему выдают личный ящик, который закрепляют за ним на ближайший год (или больше) и на котором указывают его имя.
* Когда вы приходите в бассейн, ваш шкафчик не подписывают, и он вам выдаётся на один «сеанс».

Таким образом **классические системы мониторинга думают, что они детский сад**, а не бассейн: они предполагают, что объект для мониторинга к ним пришёл навсегда или надолго, и выдают им шкафчики соответствующим образом. Но реалии в Kubernetes иные: pod пришёл в бассейн (т.е. был создан), поплавал в нём (до нового деплоя) и ушёл (был уничтожен) — всё это происходит быстро и регулярно. Таким образом, система мониторинга должна понимать, что объекты, за которыми она следит, живут короткую жизнь, и должна уметь полностью о нём забывать в нужный момент.

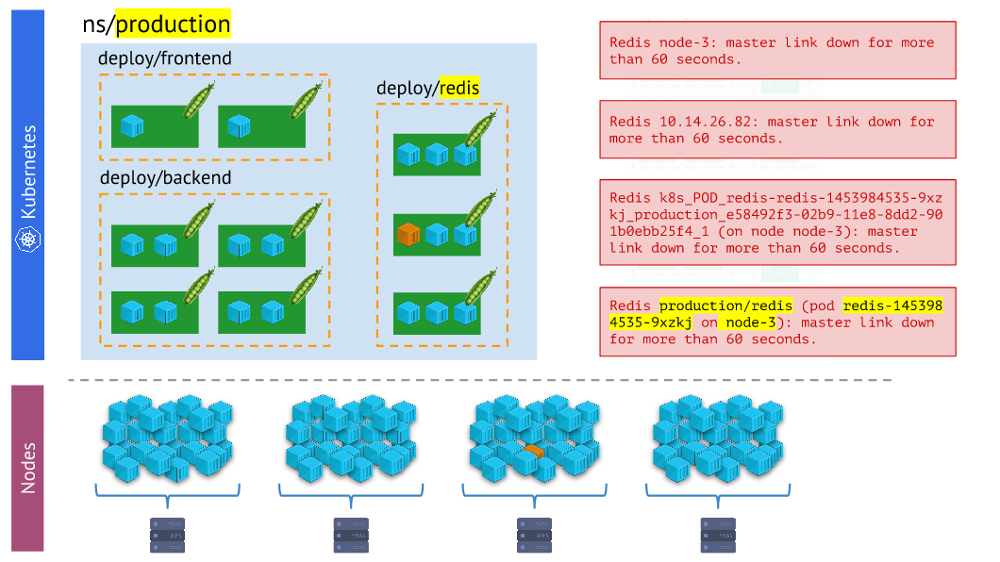
2. Параллельная реальность существует

Другой важный момент — с появлением Kubernetes у нас одновременно существуют две «реальности»:

1. Мир Kubernetes, в котором есть namespaces, deployments, pods, контейнеры. Это мир сложный, но он логичный, структурированный.
2. «Физический» мир, состоящий из множества (буквально — кучи) контейнеров на каждом узле (Рис. 4).

  
Рис. 4. Высокоуровневая модель развёртывания контейнера в кластере

И в процессе мониторинга нам необходимо постоянно **сопоставлять физический мир контейнеров с реальностью Kubernetes**. Например, когда мы смотрим на какое-то пространство имён, мы хотим знать, где находятся все его контейнеры (или контейнеры одного из его подов). Без этого алерты не будут наглядными и удобными в использовании — ведь нам важно понимать, о каких объектах они сообщают (Рис. 5).

  
Рис. 5. Разные варианты алертов — последний нагляднее и удобнее в работе, чем остальные

**Выводы**

1. Система мониторинга должна использовать встроенные примитивы Kubernetes.
2. В одном кластере, как правило, несколько окружений (помимо production), а значит — это нужно учитывать (например, не получать ночью алерты о проблемах на dev).

# **Список литературы**

1. Википедия. Kubernetes*.* URL*:* https://ru.wikipedia.org/wiki/Kubernetes (дата обращения 1.15.2018).
2. Abhishek Verma; Luis Pedrosa; Madhukar R. Korupolu; David Oppenheimer; Eric Tune; John Wilkes (April 21–24, 2015). “Large-scale cluster management at Google with Borg”. *Proceedings of the European Conference on Computer Systems (EuroSys)* (дата обращения 1.15.2018).
3. Википедия. Apache Spark*.* URL*:* https://ru.wikipedia.org/wiki/ Apache\_Spark (дата обращения 1.15.2018).
4. С. Риза, У. Лезерсон, Ш. Оуэн, Д. Уиллс. *Spark для профессионалов: современные паттерны обработки больших.* СПб, Питер, 2017. 272 с.