Довольно простой для системных администраторов концепт, довольно сложный – для меня, как для разработчика, концепт – это Service mesh в Kubernetes. Соответственно, как работает сетевое взаимодействие в Kubernetes-е по умолчанию? У вас есть node-а. На node у вас есть kube-proxy – это своего рода daemon set, который обеспечивает сетевое взаимодействие между node-ами и, соответственно, какое-то load balancing, какой-то балансировщик нагрузки.

Соответственно, у вас есть предопределенный вами набор правил для этих самых kube-proxy. Вы говорите, что у вас есть вот такие вот сервисы, Pod-ы принадлежат таким-то сервисам, и такие-то запросы идут в такие-то сервисы. Ну, кто занимается системным администрированием, вспомнили правила API tables, это примерно то же самое. И работает примерно таким же образом. И kube-proxy, соответственно, фарвардит получать эти правила, вернее, не фарвардит, он получает эти правила с Kubernetes-а API Server-а. И уже по ним работает с вашими Pod-ами. Это круто. Ну, в смысле круто? Это работает, но это – не гибко. Поэтому, соответственно, существуют различные Service mesh-ы, которые работают на уровне каждого Pod-а. К Pod-у пристыковывают свои proxy. И позволяют вам очень гибко регулировать. То есть они могут позволять вам ограничить взаимодействие между Pod-ами, то есть перероутить трафик между Pod-ами. Сделать какой-то балансировщик нагрузки для внутренних коммуникаций.

Kube-proxy – он, ну, он простой. Потому что в Kubernetes в общем не ставится основной задачей обеспечение сетевого взаимодействия. Его задача – предоставить вам декларативный механизм для описания вашей инфраструктуры приложений. А различные сертификаты, безопасность, балансировки нагрузки – это уже задача ваша, ну, или в данном случае задача так называемых Service mesh решений. То есть решений, которые вы установите в вашу систему, накидываете им каких-то дополнительных правил. И у вас это всё дело работает.

Соответственно, зачем нужен этот Service mesh? Во-первых, мы можем при настройке сети обеспечивать полную гранулярность до уровня непосредственно Pod-а. Мы можем говорить, что вот в этом ReplicaSet-е, например, у нас три Pod-а, на два из них нужно давать высокую нагрузку, на третий нужно давать нагрузку пониже. Зачем надо? Не знаю. Ну, может, пригодится.

Очень гибко, как правило, это всё настраивается. То есть помимо правила API tables, мы еще можем говорить, делать rerouting-и, мы можем делать (НРЗ 02:51), в том числе внутри сети. Мы можем делать отключение нагрузки. Мы можем делать redistribution. В общем, можем управлять сетевыми потоками вообще, как угодно. Плюс к этому, мы еще их и видим. Например, у того же istio есть frontend, называется он Kiali, насколько я помню. И он позволяет вам еще и визуально видеть, где у вас там трафик, куда он течет, по каким сервисам и так далее. И как раз это попадает в категорию дополнительных фич. То есть сетевое взаимодействие в Kubernetes-е – это очень базовая вещь. И если вам нужны какие-то функции поверх, какие-то удобные функции, а не просто API tables, то, соответственно, вы обеспечиваете там Service mesh.

Service mesh. При чем здесь golang? Зачем нам нужен golang? Но основным…, вернее, очень популярным решением для service mesh-а является istio. Я до сих пор не знаю, как это правильно произносится. Вообще, с этими сетевыми движками какие-то проблемы с наименованиями, что inject, что istio. В общем, она написана на golang-е. И ее расширения, если вам таковые нужны, тоже пишутся нативно на golang-е. Это хороший аргумент, на мой взгляд.

И основные тулзы service mesh также написаны на GO. Фишка в том, что как раз… Опять же, тот мой самый любимый аргумент про минимум ресурсов, необходимых для proxy здесь работает практически идеально, потому что вам не нужно никаких

библиотек устанавливать в ваши proxy-контейнеры для того, чтобы они работали. Достаточно основного приложения и каких-то системных библиотек.

Давайте установим istio, посмотрим на то, что istio делает с Kubernetes кластером для обеспечения service mesh-а. То есть для того, чтобы нам обеспечить Routing между сервисами с какими-то правилами, сбор метрик и защиту, то есть защищенность внутри нашего кластера, а не только снаружи. И начнем мы с того, что будем это самое istio устанавливать.

На, соответственно, сайте с описанием istio мы видим, что у istio есть свой Control plane. Точно так же, как у Kubernetes-а. То есть это такой набор Pod-ов, которые обеспечивают как раз Routing, которые сканируют наши, значит, Pod-ы. И понимают куда нам нужно ставить proxy. Напоминаю, что proxy – это такой механизм istio, который позволяет сервисам не думать о том, как происходит коммуникация, как нам обеспечивать безопасность коммуникации друг с другом, как ограничивать коммуникацию сервисов друг с другом внутри кластера. Это все делает istio. И для этого istio инжектит, то есть создает внутри Pod-а свои proxy. И эти самые proxy обеспечивают сетевое взаимодействие. То есть все данные, которыми ваш сервис обменивается в интернете, то есть внутри кластера в сети, они идут на proxy. И proxy уже думают там: куда их, значит, перенаправить, можно ли их перенаправлять или нет, каким образом их перенаправлять этим всем. И это вся информация содержится внутри proxy. И поступает оттуда с помощью control plane из Istiod.

Если мы сейчас откроем наш Kubernetes кластер, мы увидим, что никакого Istiod у нас ни в каких, значит, namespace-ах у нас вообще-то нету: ни в kube-puplic, ни в kube-system, ни уж тем более в, вот тут вот мы видим, что ничего такого, даже похожего на Istio у нас нету. То есть Istio у нас в кластере сейчас нет, его надо, естественно, устанавливать. Вот этим мы сейчас и займёмся.

На официальном веб сайте у нас есть несколько путей установки Istio. Первый путь — это установка приложения Istio сети на вашу машину. И установка с помощью консоли, с помощью этой консольной сети. Второй путь — это установка с помощью Helm-а. Helm — это приложение, которое позволяет вам, так сказать, автоматизировать установку приложений в Kubernetes. То есть не просто оперировать вот такими вот yml chart-ами, а ещё и их разделять, автоматически обновлять. Ну, в общем, ведёт себя как пакетный менеджер для Kubernetes-а.

Мультикластерная установка. External Control Plane установка, когда у вас несколько кластеров. Установка на виртуальную машину и установка оператором, которая в общем-то была (НРЗ 07:48).

Мы воспользуемся первым путем — установкой с помощью Istioctl. Здесь написано, что, значит, нам нужно скачать Istio release. Istio release для Linux or macOS качается вот таким образом с помощью curl. Для «Винды» я просто взял, да и скачал архив. Можно скачать только Istioctl, а можно скачать архивчик, который ещё содержит всякие демки и всякие дополнительные вещи. Я рекомендую скачать архив со всем, потому что мы им воспользуемся в дальнейшем. В папке, которую мы распаковали, есть соответственно, исполняемый файлик Istioctl. Вы можете его добавить в path. Я его скопировал себе, соответственно, в folder в свою папочку, в которой я буду работать. И отсюда мы можем запустить Istioctl. И он нам выдаст то, что он умеет. Сейчас нас, соответственно, интересует команда из Istio install. Istioctl install, который установит к нам Istio с профилем по умолчанию в наш Kubernetes-овский кластер.

Итак, из Istio у нас установлено. Опять же, если у вас, например, minikube. То есть вы работаете на системе mac, никакой разницы у вас не будет. Istio ctl найдет по вашему профилю, найдёт подходящий kubernetes и туда поставит Istio. И теперь при получении namespace-ов, мы видим namespace Istio system, в котором у нас

есть ingressgateway. То есть это и Istio-чный контроллер для того, чтобы входящие запросы обрабатывать и перенаправлять, как надо. И Istiod, вот этот самый Control plane, который у нас инжектит наши proxy, который обеспечивает этим proxy-ям доставку информации и всё прочее.

Ну теперь давайте посмотрим, как Istio работает. Для тестирования нашего Istio сервиса, я взял пример микросервисов с google cloud. То есть там есть у них примеры всякого различного, различных сервисов для того, чтобы посмотреть, как это всё будет деплоиться и работать в kubernetes-е. На самом деле, здесь всё очень просто. Здесь в основном deployment-ы с различными сервисами. И сами kubernetes-овские сервисы, которые этим микросервисам определяют, соответственно, порты и пути для того, чтобы с ними общаться. Естественно, у нас тут сервис через сервис, но здесь, к сожалению, терминология уже устоялась. То есть, есть deployment-ы kubernetes-овские и есть сервисы, которые эти deployment-ы связывают и обеспечивают какой-то outing. Мы это всё задеплоим в наш кластер, заинджектим в Istio. И посмотрим, как это работает с точки зрения администратора кластера.

Однако если мы сейчас загрузим в наш дефолтный кластер, загрузим наш сервис, у нас ничего не сработает, потому что дефолтный кластер должен иметь Label для того, чтобы Istio уже знал, что ему делать. Поэтому создадим Label. Label namespace. Вернее, не кластер, а естественно, namespace, наш дефолтный namespace. Значит создаём namespace default. И делаем Label Istio Inject. Injection enabled. Соответственно, теперь у нас есть Label, мы его можем увидеть, если мы пишем наш namespace. То есть Label-ы видны, естественно, контроллерам внешним, которые опрашивают namespace-ы — это есть в kubernetes-овском API.

И мы теперь можем сделать kubectl. Зайти в папочку kubernetes. И сделать kubectl apply -F. Наш сервис mesh.yaml. И у нас создались наши pod-ы. И можно видеть,

что несмотря на то, что в описаниях описан обычно один контейнер, то есть вот он один контейнер, и так почти везде, создаются у нас здесь контейнеров 2. То есть в момент создания, значит, pod а у нас есть Istio Control plane, который знает, что у этого namespace-а включен Label Istio injectable. И создает к нему, соответственно, Invoice Proxy, то есть ту самую Istio-шную Proxy — вот она здесь. Соответственно, здесь все эти Proxy. У Proxy есть куча настроек, эти настройки менеджерятся Istio-шным, соответственно, Control plane-ом, они к нашему приложению не относятся и это чудесно. Потому что разработчики приложения? они должны думать, да? как там что это такое Istio, что это такое сервис mesh. Они просто должны сделать приложение, его задеплоить, как-то общаться с сетью, а всё остальное должны на себя взять, соответственно, наше Istio. Ыключая безопасность, включая routing. И включая анализ и сбор логов, в том числе. Значит, у нас должен был быть уже, да, у нас уже поднялись все, в принципе, контейнеры, есть вот генератор, который генерирует трафик внутри сети.

А теперь давайте установим инструменты мониторинга Istio и посмотрим, что же там происходит. В папочке Istio samples addons у нас есть некоторые интеграции с Istio. То есть Istio, одна из задач Istio — это предоставлять телеметрию и какие-то метрики. Для этого Istio пользуется Prometheus-ом для сбора метрик. И соответственно, отображения их в Graphane-е, который тоже поставляется вместе с Istio. Ну, то есть можно её установить. Есть eger, который собирает (НРЗ 14:23), то есть собирает путь запросов внутри вашей микросервисной архитектуры, может отслеживать ваши распределённые транзакции и прочее — очень полезная тулза.

И есть панелька для отображения вашей сетевой активности kiali frontend. Давайте мы их установим. Соответственно, есть папочка extras, с которой ещё устанавливается zipkin, который является конкурентом eger-а. И всякие дополнительные addon-ы для Prometheus-а в виртуальных машинах, поддержки tls-а, оператор Prometheus-а — это всё нам не нужно. Поэтому я копирую только 4

yaml файла, вставляю их, опять же, в свой folder Istio addons. применяю его в нашем, на наш кластер. Соответственно, у kubectl команда apply работает не только на файлы, она также может работать на folder. И установит просто оттуда все файлы. Если вот так сейчас нажму, то у нас всё установится и Graphane-a, и eger. И соответственно, Kiali. И Prometheus. То есть мы всё установили, подождём, пока всё загрузится, и сейчас я вам покажу фишку, которая мне очень нравится в Istio.

Итак, у нас всё установилось. У нас теперь в папочке namespace Istio system есть pod Graphane-a, есть pod eger-a, есть Kiali и Prometheus. И также у них есть свои сервисы для того, чтобы, например, отображать frontend-ы в случае с Graphane-ой. И отображать frontend в случае с Kiali.

Давайте сделаем Port Forward сервиса Kiali. И посмотрим, что там внутри. Port Forward – это непосредственно mapping из kubernetes-овского кластера, прокидывание всех пакетов определённого протокола на Port нашей машины. То есть мы можем смапить любую машину на свой localhost. И давайте попробуем это сделать. Сделаем Port Forward. В данном случае мы мапим сервис, поэтому у нас будет svc kiali. И дальше указываем Port. Если вы укажете вот так, то смапите удалённый порт 2000…, 20001, на ваш локальный порт 8080. Я просто смаплю Port 2001 на мой Port 8080. И посмотрим, что же там сейчас внутри. (НРЗ 16:58) откроем, откроем наш localhost 20001. И это будет консоль kiali.

Здесь есть много чего прикольного, здесь есть workload. Как бы что, значит, у нас происходит, какой статус здоровья у серверов, подняты они или нет. У меня у кластера довольно мало ресурсов выделено, поэтому тут кое-какие вещи периодически падают. Их очень хороший. Очень хороший пример, как раз как, как не должно быть в production-е.

И что самое прикольное есть график. Есть график того, как сервисы взаимодействуют друг с другом. Здесь надо ещё покликать, чтоб найти удобное местоположение для себя. То есть можно увидеть, да, что у нас здесь есть. Вот генератор, который генерирует всю нагрузку, и у вас может быть это Ingress Controller, будет Ingress Controller в ваших внешних сетях, то есть контроллер, который извне обрабатывает запросы. Он идёт на frontend-ы, frontend-ы обращаются к различным сервисам внутри сети по протоколам grpc. Там сервисы обращаются к базам данных по протоколам TCP. В общем, всё видно.

Если вы с помощью kiali, с помощью Istio делаете так называемые (HP3 18:13), когда у вас есть несколько версий сервисов, здесь у вас будут несколько, простите…, здесь у вас будут несколько версий различных, вы сможете с помощью его настроек Istio управлять процентовкой. То есть вы можете говорить, что 90% трафика идёт ещё на старый сервис, 10% идёт на новый сервис. Довольно удобно. Здесь будет всё видно: что плохо и что хорошо, какой процент у нас, соответственно, успехов и ошибок, какого типа трафик преобладает и так далее. И какие ещё ответы. Очень удобно то, чтобы одним взглядом окинуть свою инфраструктуру и что-то понять.

Опять же, моя цель здесь — просто показать, как Istio работает с самим Kubernetes-ом, каким образом Istio сам по себе сервис mesh устанавливает свой control plane, что вы в pod-ы можете загружать любые namespace. Это не гайд по Istio, это очень…, ну это даже не база, это то, что можно найти прям в демо документе. Поэтому, если вы хотите прокачаться в сервис mesh, надо будет, ну соответственно, посмотреть пару видео по сервис mesh, поиграться на вашем кластере, посмотреть, как это работает, узнать про сервис центр Istio-шный, про виртуал сервис Istio-шный. И просто много экспериментировать.