Что стоит знать о безопасности и управлении доступом в Kubernetes

Не забудь включить запись!

План

- Безопасность Docker на хост-машине
- Нужные примитивы и понятия
- Сетевая безопасность
- Секреты и их хранение
- Практические рекомендации

Безопасность Docker на хост-машине

- Про это будет выделенное занятие, я лишь повторю очевидное
  - Использовать нормальную ОС
    - CoreOS
    - Google Container Optimized OS

> Fedora Atomic Host теперь вливается в дружную семью

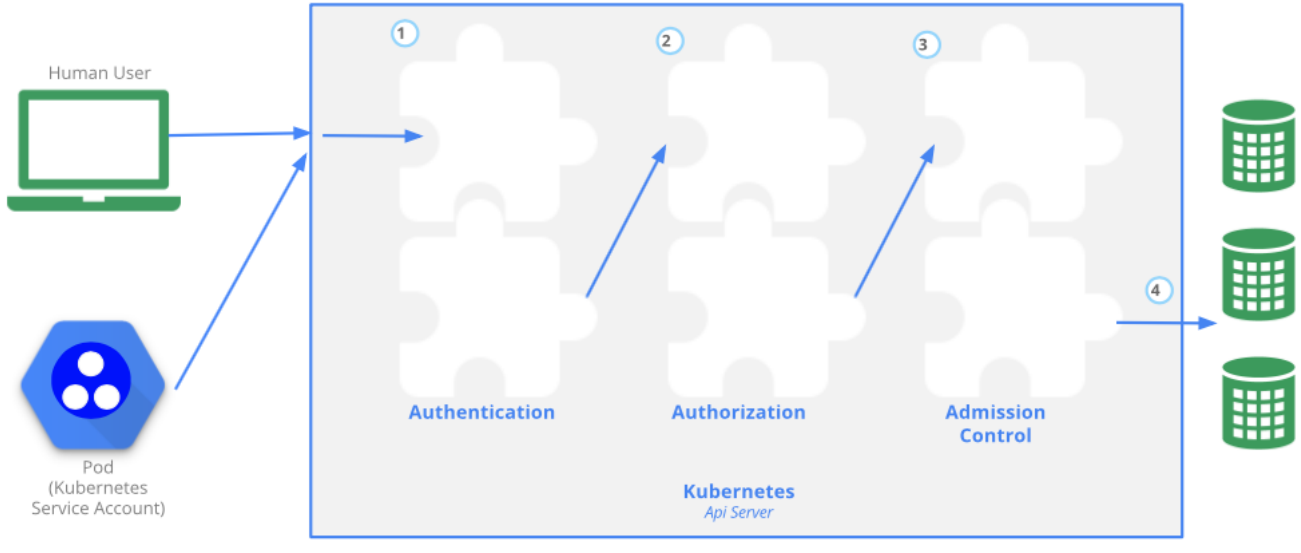
Безопасность Docker на хост-машине

- Ограничивать внешнюю связность машин из кластера
  - Выделенный роутер для этих задач
  - Не давать им белый IP
- Использовать готовые инструменты для анализа безопасности
  - CIS Kubernetes Benchmark
  - CIS Docker Benchmark
  - kube-bench
- hostPID опасная штука (если есть SYS_PTRACE capability)
- Голову никто не отменял!

Нужные примитивы и понятия

AAA против AA

Помним о том, что k8s разделяет эти понятия (и правильно делает):



- Аутентификация = ответ на вопрос "кто ты?"
- Авторизация = ответ на вопрос "что тебе можно?"
- Admission Controllers (AC) = дополнительный контроль и опции

Аутентификация

Виды пользователей

- Обычные пользователи
  - Это люди, которые отдают команды кластеру
  - Глобальны в рамках кластера
  - Не управляются из API
- Service Accounts
  - Это процессы, которые работают внутри кластера и тоже общаются с API
  - Локальны в Namespace
  - Управляются из API
  - Привязаны к токену из Secrets, позволяют элементам кластера общаться с API

"Не управляются" это как?

Вы должны каким-то образом объяснить k8s, откуда аутентифицировать:

- X509 Client Certs
- Static Token File / Static Password File
- Bootstrap Tokens / Service Account Tokens
- OpenID Connect Tokens
- Webhook Token Authentication
- Authenticating Proxy
- Анонимный запрос, урашечки! 🤓

С небес на землю

А что реально используется?

- X509 Client Certs
  - k8s не поддерживает отзыв сертификатов
  - Можно делать их короткоживущими, но это много телодвижений
- Static Password File
  - Для маленького окружения или dev-окружения
  - Для agile "Один за всех и все за одного"
- OpenID Connect Tokens
  - Внешняя система рулит этим вопросом

А в моем кластере?

- Помним, что способы аутентификации задаются при помощи параметров командрой строки api-server
  - Формально нужно ее посмотреть и понять, что там
- К размышлению:
  - `--basic-auth-file=string`
  - `--token-auth-file=string`
  - `--anonymous-auth=true`

Static Password File

Demo 1

Теперь о контекстах

- От того, что мы создали пользователя в кластере, ничего не изменилось
- Нужно как-то попробовать зайти от его имени, наверное?
- Пилим `~/.kube/config` + `KUBECONFIG`

Меняем контекст и пробуем

Demo 2

Выводы

- Судя по всему, аутентификация нам удалась
- Не очень удалась авторизация, то есть нам нельзя делать то, что мы хотим

> Самое время разобраться, а как же я работал с кластером до этого? Но чуть позже.

Service Account

- Аккаунты пользователей - для пользователей, то есть для людей
  - Service Account для процессов в Pod
- Аккаунты пользователей глобальны в рамках кластера
  - Service Accounts живут в рамках namespace (это удобно для разных приложений внутри кластера)

Service Account

- Аккаунты пользователей обычно должны произрастать из бизнес-среды, где создание аккаунта есть некий процесс
  - Service Accounts более легковесны и легко создаются под нужную задачу, лучше подходя под принцип наименьших привилегий
- Аудит для аккаунтов пользователей и Service Accounts может быть сильно разный

Создаем Service Account

Demo 3

Авторизация

А что дают?

- Существует 4 модуля авторизации:
  - Node
  - ABAC
  - RBAC
  - Webhook

А что у меня в кластере?

- `kubectl cluster-info dump | grep authorization-mode`

```
--authorization-mode=Node,RBAC
```

- и так тоже можно
  - `--authorization-mode=AlwaysAllow`
  - `--authorization-mode=AlwaysDeny`

Node

Этот модуль дает возможность kubelet совершать определенные действия с API, то есть:

Читать:

- Services
- Endpoints
- Nodes
- Pods
- Secrets, ConfigMaps, persistent volume claims, persistent volumes которые относятся к Pod в этой Node

Node

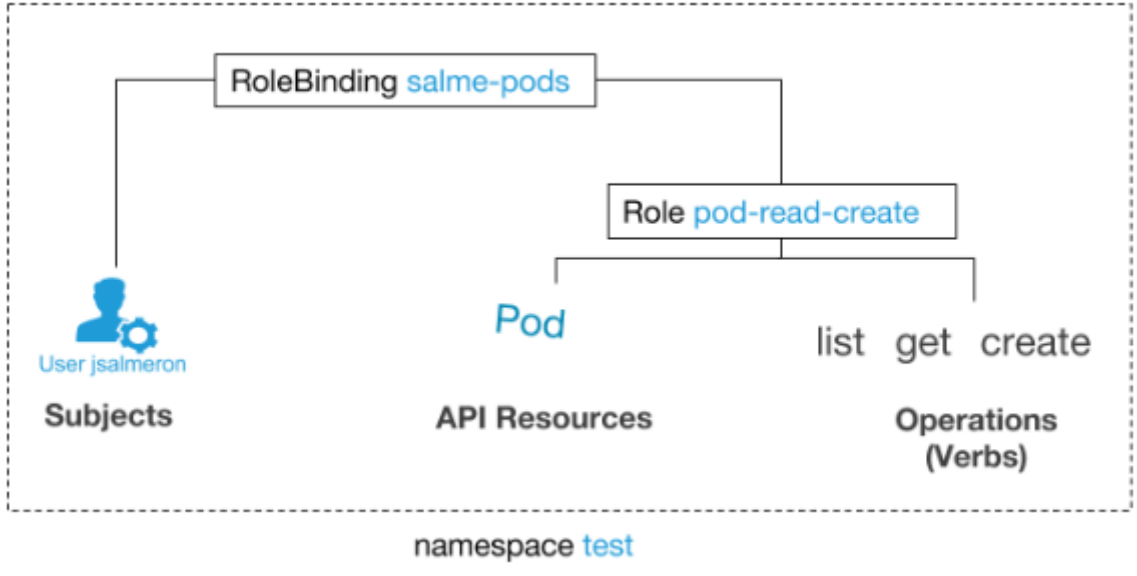
Записывать:

- Nodes и их статус
- Pods их статус
- События

NodeRestriction бежит на помощь в ограничении власти kubelet при помощи этого модуля

RBAC

Субъект + (Операция + Ресурс)



Операция + Ресурс = Role

Субъект <-> Роль = RoleBinding

ABAC

- Attribute-based access control (ABAC) строится на парадигме, где права доступа выдаются через использование политик
- Политика связывает вместе права и набор атрибутов
  - `{"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}`
  - `{"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readonly": true}`

Webhook модуль

- apiserver нужно передать вот этот параметр с путем к файлу настроек
  - Куда ломиться для авторизации хотелок пользователя
  - `--authorization-webhook-config-file string`

Отсюда и далее: RBAC

Роли

- Начнем с ролей, которые по сути своей связывают набор операций и ресурсов.
- Роли могут быть ограничены namespace (Role), так и простираться на весь кластер (ClusterRole)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" означает apiGroup под именем core или legacy
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Роли

- Пожалуй, самая мякотка тут в apiGroups, то есть в том, к чему мы даем доступ
- Параметр `namespace` не применяется в ClusterRole по понятным причинам

apiGroups

- core
- batch
- apps
- extensions

Нет простого списка, поскольку сейчас главная задача разработчиков растащить core на что-то более вменяемое

Идем ножками в документацию и смотрим

- `kubectl api-resources` покажет нам и apiGroup и имена ресурсов

Хорошо, как теперь эту роль применить?

- Роль "накатывается" при помощи RoleBinding (или ClusterRoleBinding)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods      # под этим именем мы потом увидим этот RoleBinding
  namespace: default
subjects:
- kind: User           # Group, ServiceAccount
  name: jane           # имя чувствительно к регистру
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role           # явно указываем Role или ClusterRole
  name: pod-reader     # а тут имя той Role или ClusterRole к которой мы биндимся
  apiGroup: rbac.authorization.k8s.io
```

И еще немного примеров

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
```

```
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

```
subjects:
- kind: Group
  name: system:serviceaccounts:qa
  apiGroup: rbac.authorization.k8s.io
```

Вернемся к нашим баранам

- Мы до сих пор так и не знаем пока, почему созданный для нас пользователь в minikube работает
- И почему не работают наши два созданных пользователя

Разбираем по кускам

Demo 4

Вся власть admin!

Demo 5

Промежуточные выводы:

- Для сертификатов имеет значение поле Organization
  - `"/CN=jbeda/O=app1/O=app2"`
- В статических файлах будь то пароли или токены мы даем список групп
- А что там с Service Accounts?

Вся власть service-admin

Demo 6

Остальные роли?

- Мы пока посмотрели только роль cluster-admin, но из коробки их чуть больше
  - admin
  - edit
  - view
- Они по-умолчанию никому не выданы, их анализ — самостоятельная работа
- Ну и создавать свои роли никто, разумеется, не мешает :)

Namespaces

- Позволяют разруливать видимость объектов в кластере
  - То есть, в одном физическом кластере может быть несколько виртуальных
- Не все вещи уходят в namespaces (nodes, persistentVolumes, например, нет)
  - `kubectl api-resources` покажет в колонке namespaced - уходит ли оно в Namespace

Namespaces

- RoleBinding существует в рамках Namespace
  - То есть пользователь может обладать разными правами в разных Namespaces
- Namespaces не касаются сети, то есть, Pods видят друг друга
  - Это позволяет делать интересные вещи

Namespaces

- Prometeus с выдачей для него общих прав на весь кластер
  - Создаем отдельную ClusterRole, где описываем, куда ему можно
  - Создаем Service Account для Prometeus
  - Создаем Namespace для Prometeus
  - Создаем ClusterRoleBinding = ClusterRole + ServiceAccount

Namespaces

- Также можно использовать Namespaces для создания в нем аккаунтов с нужными правами
- Права через RoleBinding мы распределили, поэтому каждый Service Account в Namespace их получает

Имена сервисов

- В пределах одного Namespace можно писать имя
- Но вообще имена сервисов идут в виде `<service>.<namespace>.svc.cluster.local`
  - Можно обращаться, допустим, `database.staging` или `database.dev`

Namespaces

Demo 7

Admission Controllers

Admission Controller (AC)

- k8s предлагает еще одну функцию в ограничении доступа к API
- AC может делать две важные функции:
  - Изменять запросы к API (JSON Patch)
  - Пропускать или отклонять запросы к API
- Каждый контроллер может делать обе вещи, если захочет

Admission Controller (AC)

- Какие-то AC включены сразу же из коробки
- А как это посмотреть?
  - `kubectl cluster-info dump | grep enable-admission`

NamespaceLifecycle

- Запрещает создавать новые объекты в удаляемых Namespaces
- Не допускает указания несуществующих Namespaces
- Не дает удалить системные Namespaces
  - `default`
  - `kube-system`
  - `kube-public`

ResourceQuota

- Мы можем (и нужем) создавать в системе ограничения использования ресурсов
- Но следит за их выполнением именно этот AC
- Нет его? Фокус не сработает

```
apiVersion: v1
kind: ResourceQuota
metadata:
name: mem-cpu-demo
spec:
hard:
  requests.cpu: "1"
  requests.memory: 1Gi
  limits.cpu: "2"
  limits.memory: 2Gi
```

LimitRanger

- Позволяет задавать лимиты для Pod, Container, PersistentVolumeClaim
- Описывает их в виде предельного и дефолтного значения

```
apiVersion: v1
kind: LimitRange
metadata:
name: mem-limit-range
spec:
limits:
- default:
    memory: 512Mi
  defaultRequest:
    memory: 256Mi
  type: Container
```

И еще

- NodeRestriction
  - Ограничивает возможности kubelet по редактированию Node и Pod
- ServiceAccount
  - Автоматически подсовывает в Pod необходимые секреты для функционирования Service Accounts
- Mutating + Validating AdmissionWebhook
  - Позволяют внешним обработчикам вмешиваться в обработку запросов, идущих через AC

[Pod]SecurityContext

- Два объекта, которые контролируют фичи (без)опасности Container или Pod
- Свойство securityContext его содержит
- Наиболее часто используемое:
  - UID, GID
  - Возможность привелигированного запуска (доступ к устройствам)
  - Linux Capabilities 🦄
  - AppArmor и SELinux

PodSecurityPolicy

- Но отсюда сразу же возникает вопрос - как контролировать [Pod]SecurityContext в рамках кластера?
- PodSecurityPolicy позволяет нам контролировать [Pod]SecurityContext
- Его читает и применяет нужный PodSecurityPolicy AC
- Достаточно граблеопасен, поскольку в RBAC-среде требует вдумчивого анализа
  - И внимательного включения

Как это работает?

- Мы создаем PodSecurityPolicy, в которой описываем параметры Pod
- Даем право пользоваться этой PodSecurityPolicy при помощи Role + RoleBinding
- Любой создаваемый Pod должен быть одобрен AC
- Если мы в политике для пользователя сделали `privileged: false`, а он пытается сделать `privileged: true`, то Pod не создастся
- Так мы можем гранулярно решать, кому какие (не)безопасные фичи можно использовать

На практике

- Если мы без настройки включим PodSecurityPolicy AC, то мы приложим наш кластер
  - Нет ни одной политики, поэтому нельзя создавать Pod
- Поэтому сначала создаем наиболее разрешительную политику
- Всем даем право ее использовать
- Дальше начинаем создавать более запретительные политики, вкатывая их по мере необходимости

Сетевая безопасность

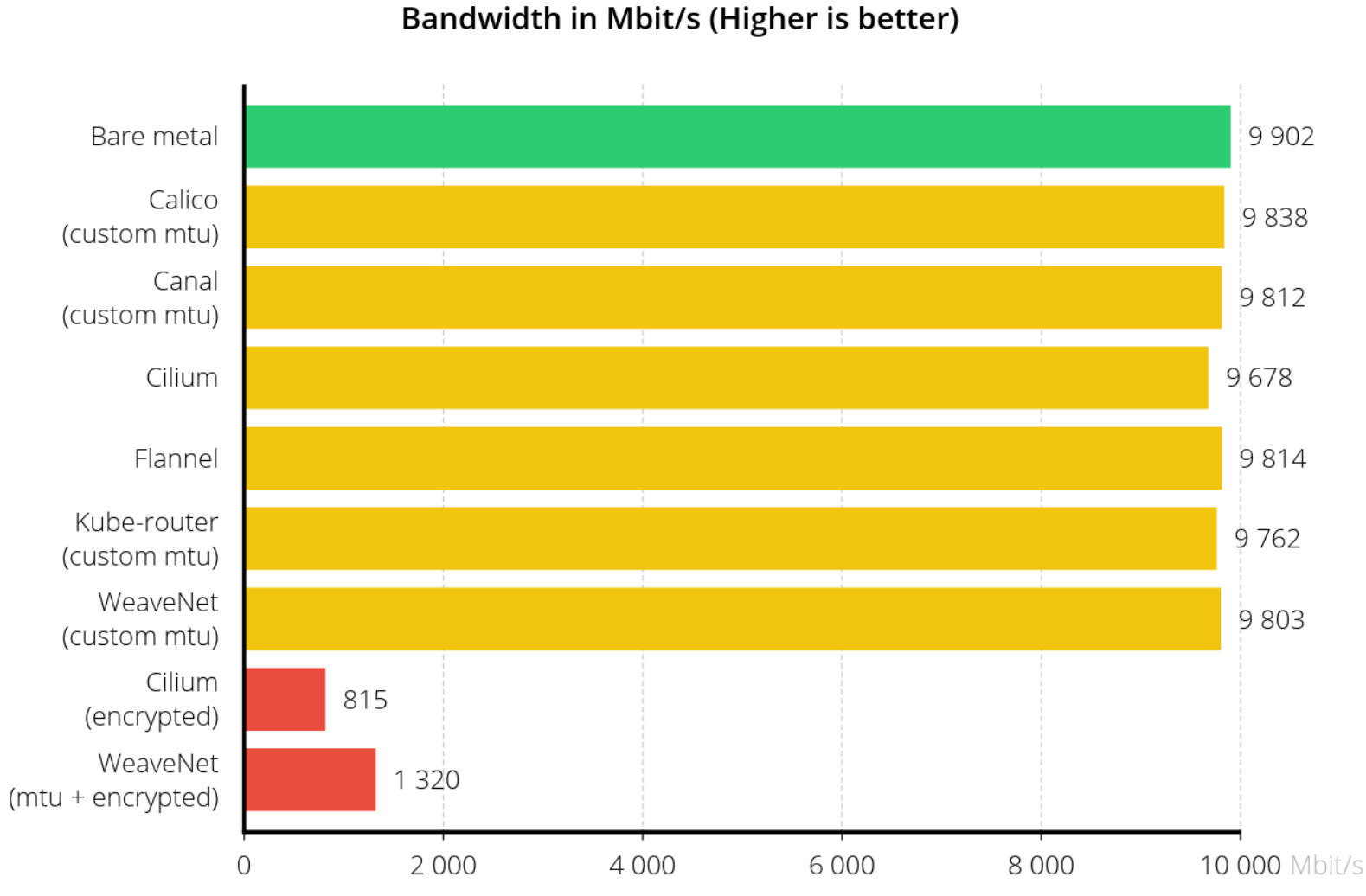
Как страшно жить

- Внутри кластера все контейнеры могут общаться между собой
  - И да, даже между разными Namespaces, кто же их остановит?
- Адреса могут динамически меняться
- Нет шифрования трафика из коробки

Как с ней жить?

- Сложно фильтровать трафик в условиях изменчивости адресов
  - NetworkPolicy решает проблему (Flannel их не поддерживает)
- Шифрование трафика дает достаточно большой оверхед

Kubernetes CNI benchmark - 10Gbit network - TCP



2019-04-05 - Alexis Ducastel - https://infrabuilder.com - Benchmark tool : iperf3

Секреты и их хранение

Что такое секреты (Secrets)?

- Пароли к базам, токены и так далее
- С одной стороны, они нужны для работы сервиса
- С другой стороны, мы не хотим их показывать всем и вся

Есть два общих подхода

- Либо хранить их в специальном объекте Secret
  - Не шифруются, хранятся в Base64
  - Начиная с 1.13 можно шифровать
- Либо использовать Hashicorp Vault для этого (будет занятие по этой теме)

Практические рекомендации

Доступ разных команд

- Namespaces + RBAC = наше всё
- По большей части, разница есть только в размере команд
- Разумеется, мы подразумеваем нормальные DevOps практики
  - Кодом делимся через git, пишем нормальный инфра-код

Размер имеет значение

- Маленькой команде (5-10 микросервисов) вряд ли нужен Namespace для разработки, они могут катать Minikube
- Средней команде (10+ микросервисов) уже обязательно использовать разные Namespace, но несколько кластеров пока не нужно
- Крупным командам, где локально запустить проект невозможно, уже есть смысл использовать несколько разных кластеров

Уровни доступа

- Разработчики не имеют доступа к production-окружению
  - Read-only если это так необходимо и только к ряду ресурсов
- В ряде случаев предполагается выделение роли релиз-инженера, который отвечает за деплой
  - Тогда разработчиков даже в staging-окружение пускают только для чтения

Сегодня мы многое поняли 🧐

- Посмотрели, как устроен стек AAA в k8s
- Разобрали, как мы авторизуемся и как управлять контекстами
- Понимаем, что такое RBAC и зачем оно нам нужно
- Знаем, что нужно использовать Namespaces для деления кластера на кусочки