Kubernetes 集群

课程目标

- 理解 Kubernetes 集群负载均衡
- 理解 Kubernetes 自动伸缩的概念
- 理解 Kubernetes 常用的调度方式
- 了解 Kubernetes 高可用的架构
- 理解 Kubernetes 的安全管理

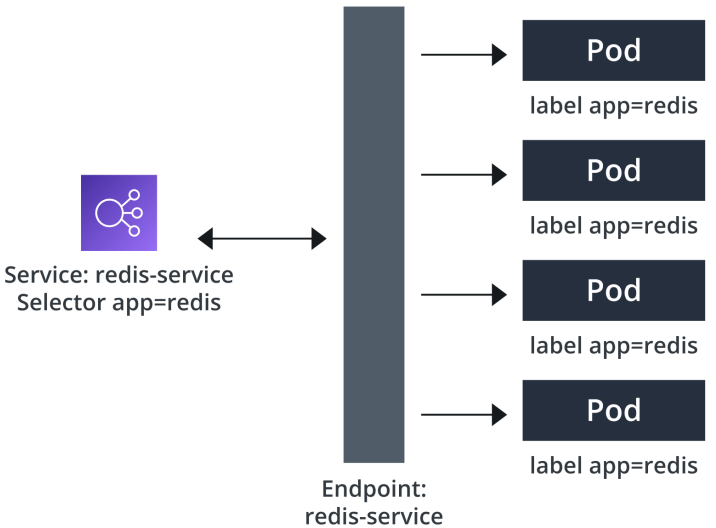
负载均衡

介绍

- 服务负载均衡：使用服务对象将请求分发到标签相同的Pod。默认使用轮询或基于IP的负载均衡策略。
- 外部负载均衡：与云服务商集成，通过负载均衡器将外部流量转发到集群中的服务。
- Ingress控制器：配置 HTTP/HTTPS 流量路由规则，把外部的流量路由到不同的服务。可以集成不同负载均衡器，如Nginx、Traefik等。

负载均衡

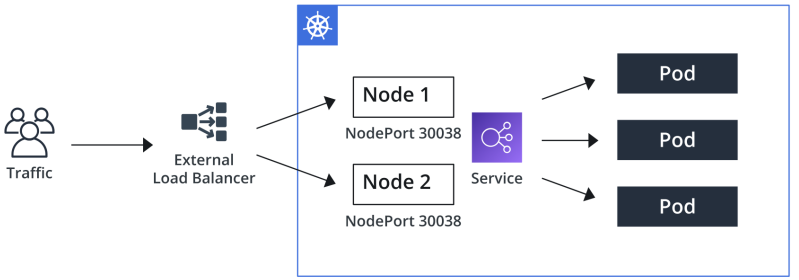
服务负载均衡



- Kubernetes中的服务对象代表一组Pod，并提供稳定的虚拟IP（Cluster IP）作为访问入口。
- 内置的负载均衡器（如kube-proxy）将请求均衡地分发到服务关联的后端Pod。
- 可以使用不同的负载均衡策略，如轮询、会话保持或基于IP的负载均衡。
- 服务负载均衡适用于集群内部流量，对于外部流量可结合使用外部负载均衡器或Ingress控制器。

负载均衡

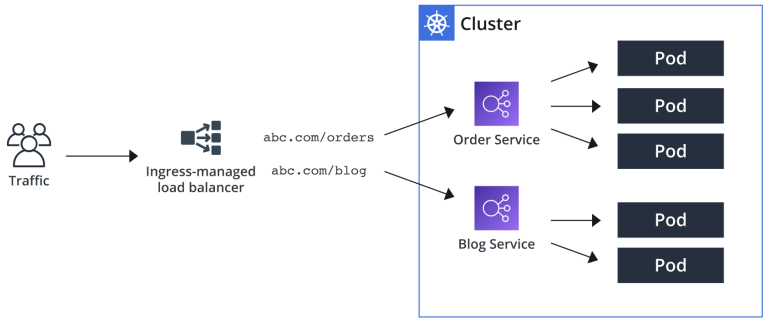
外部负载均衡



许多云提供商（如 AWS、Azure、Google Cloud 等）都提供了托管的负载均衡服务。通过在云平台上配置和使用这些负载均衡器，可以将外部流量分发到 Kubernetes 集群中的 Service。

负载均衡

Ingress控制器



它负责处理 Ingress 对象，并根据定义的规则将外部流量路由到集群内部的 Service。

- 路由和流量转发：Ingress 控制器基于 Ingress 规则来路由和转发外部流量。Ingress 规则定义了访问集群内服务的规则，可以根据域名、路径、TLS 加密等进行流量转发和策略配置。
- 外部负载均衡：Ingress 控制器通常与负载均衡器配合使用，将外部流量均衡地分发到后端的 Service 或 Pod。负载均衡器可以是云提供商的负载均衡服务或软件负载均衡器，具体取决于部署环境。
- TLS 加密和安全性：Ingress 控制器支持通过 TLS（Transport Layer Security）进行流量的加密和安全传输。可以配置 TLS 证书和密钥，以保护外部流量的隐私和安全性。
- 多协议支持：Ingress 控制器可以支持多种协议，如 HTTP、HTTPS、TCP 和 UDP。它能够根据 Ingress 规则中定义的协议类型来处理和转发相应的流量。

负载均衡 Review

- 3种负载均衡方式：服务负载均衡、外部负载均衡、Ingress控制器。

自动伸缩

- 垂直扩展（Vertical Pod Autoscaling，VPA）：可以根据容器内资源的使用情况，自动或者手动调整容器的 CPU 和内存请求量，以优化资源利用和性能。
- 水平扩展（Horizontal Pod Autoscaling，HPA）：可以根据指标（如 CPU 使用率、内存使用率）自动调整副本数，使得应用程序能够根据负载的变化进行弹性扩容和缩容。
- 自动扩展集群（Cluster Autoscaling）：可以根据集群中的资源需求自动扩展或缩减集群的节点数。

自动伸缩

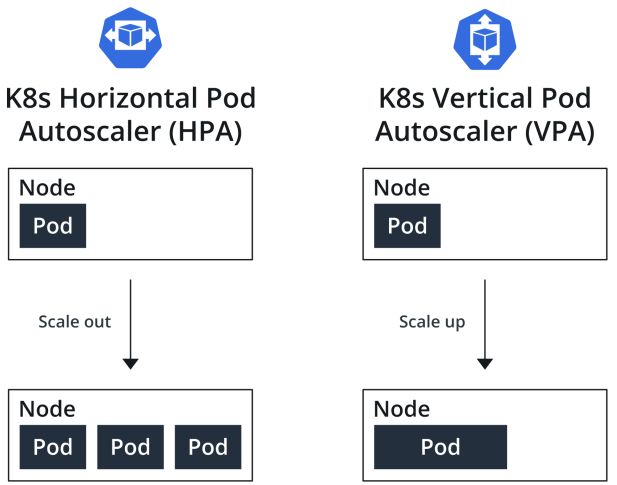
HPA 例子

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hello-deploy
  namespace: hpa-test
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hello-deploy
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

- minReplicas：最小副本数
- maxReplicas：最大副本数
- targetCPUUtilizationPercentage：目标 CPU 利用率百分比，当这个 deployment 超过这个百分比，HPA 会自动增加副本数以满足负载需求。反之减少副本数。

自动伸缩

VPA vs HPA



HPA 用于调整应用程序的副本数，以满足负载需求，而 VPA 用于调整单个 Pod 的资源配置，以优化性能和资源利用率。它们也可以一起使用，以确保应用程序在水平和垂直方向上都具有适当的资源配置。

自动伸缩 Review

- 3种自动伸缩方式：垂直扩展、水平扩展、自动扩展集群。

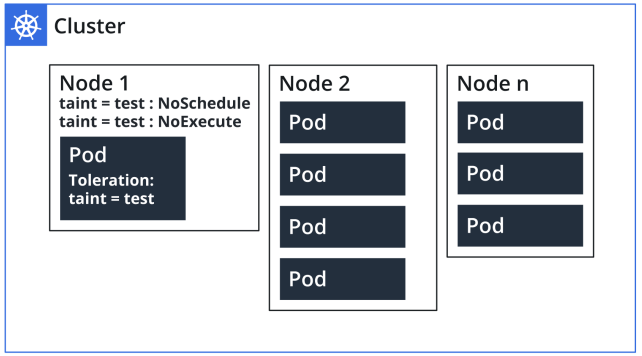
调度

调度技术

- Taints 和 Tolerations： Kubernetes 中用于节点污点和容忍的机制，用于限制哪些 Pod 可以调度到特定的节点上。
- NodeSelector：用于在调度 Pod 时选择特定的节点。
- Node Affinity：用于在调度 Pod 时指定节点的亲和性规则。通过使用 Node Affinity，可以定义一组规则来筛选适合的节点来运行 Pod。
- Pod Affinity 和 Anti-Affinity：用于指定 Pod 之间的亲和性和反亲和性规则。Pod Affinity 允许将 Pod 调度到与其他 Pod 具有相似属性或标签的节点上。Pod Anti-Affinity 则相反，允许将 Pod 调度到与其他 Pod 具有不同属性或标签的节点上。

调度

Taints 和 Tolerations

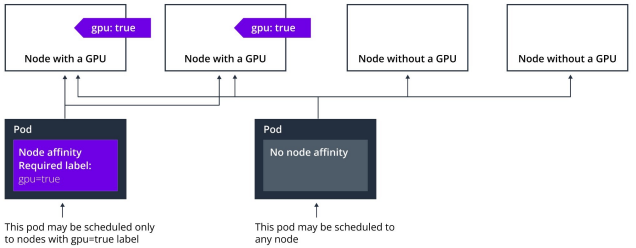



```
kubectl taint nodes nodename taint=test:NoSchedule
kubectl taint nodes nodename taint=test:NoExecute
```

- NoSchedule： 表示 Pod 在当前节点上不会被调度，但是如果没有其他可用的节点，Pod 仍然可以被调度到当前节点上。这通常是由于节点资源不足或节点标签不匹配导致的。
- NoExecute： 表示 Pod 在当前节点上不会被调度，并且如果 Pod 已经在当前节点上运行，它将被驱逐。这通常是由于节点故障、节点维护或节点被标记为不可调度导致的。

调度

Node Affinity



通过使用 Node Affinity，可以在 Pod 的配置文件中定义一组规则，以指定 Pod 对节点的偏好。这些规则可以基于节点的标签、资源和其他条件进行匹配。Node Affinity 提供了更灵活和精细的控制，可以实现更高效的调度策略。

Node Affinity 可以分为两种类型：required 和 preferred。

- Required Node Affinity：指定 Pod 必须运行在满足规则的节点上。如果没有满足规则的节点可用，Pod 将无法调度。
- Preferred Node Affinity：指定 Pod 偏好运行在满足规则的节点上，但如果没有满足规则的节点可用，Pod 仍然可以调度到其他节点上。

Node Affinity 可以用于各种场景，例如将特定类型的工作负载分配给具有特定硬件或软件配置的节点，或者将特定的 Pod 分散在不同的节点上以实现负载均衡。通过合理使用 Node Affinity，可以优化 Pod 的调度和资源利用。

调度

Node Affinity 例子

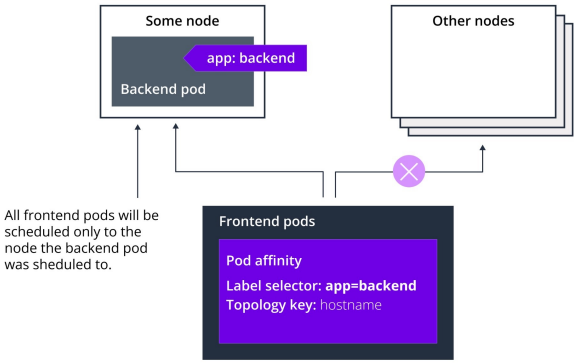
```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
  - name: gpu-container
    image: tensorflow/tensorflow:latest-gpu
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: accelerator
            operator: In
            values:
            - gpu
```

在这个示例中，我们定义了一个 Pod，它具有一个 Node Affinity 规则。规则要求在调度 Pod 时，必须满足以下条件：

- 节点的标签中必须包含 accelerator=gpu 的键值对。

调度

Pod Affinity



Pod Affinity 允许将 Pod 调度到与其他 Pod 具有相似属性或标签的节点上。通过使用 Pod Affinity，可以将具有相同功能或相互依赖的 Pod 部署到同一节点上，以提高性能和效率。例如，可以将具有相同服务或数据依赖关系的 Pod 部署到同一节点上，以减少网络延迟和数据传输时间。

调度

Pod Affinity 例子

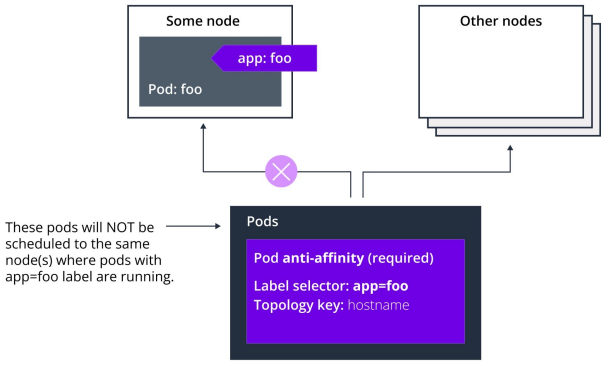
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend-pod
spec:
  containers:
  - name: frontend-container
    image: nginx
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - backend
        topologyKey: "kubernetes.io/hostname"
```

在这个示例中，我们定义了一个 Pod，它具有一个 Pod Affinity 规则。规则要求在调度 Pod 时，必须满足以下条件：

- 必须有一个具有标签 app=backend 的 Pod 在同一个节点上。
- 使用 kubernetes.io/hostname 作为拓扑键来进行调度。

调度

Pod Anti-Affinity



Pod Anti-Affinity 则相反，允许将 Pod 调度到与其他 Pod 具有不同属性或标签的节点上。通过使用 Pod Anti-Affinity，可以将具有相似功能但需要分散部署的 Pod 分配到不同的节点上，以提高容错性和可用性。例如，可以将同一应用程序的多个副本分散在不同的节点上，以防止单点故障。

调度

Pod Anti-Affinity 例子

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                    - my-app
              topologyKey: kubernetes.io/hostname
      containers:
      - name: my-app
        image: my-app-image
```

在上面的例子中，我们使用 Pod Anti-Affinity 来确保同一个 StatefulSet 中的三个副本 Pod 不会被调度到同一个节点上。 这样做可以增加应用程序的容错性，即使一个节点发生故障，其他节点上的副本仍然可以正常运行。

调度

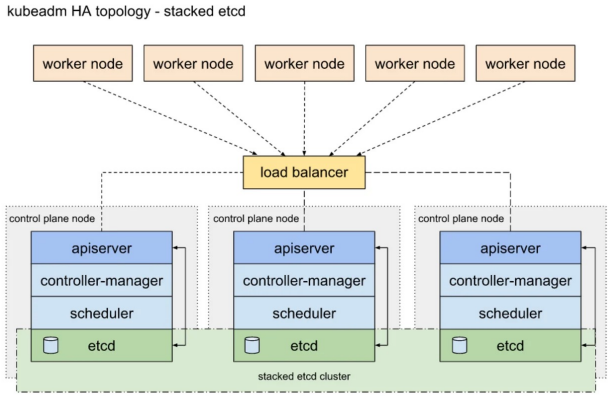
Pod Affinity 和 Node Affinity 区别

- Pod Affinity： 定义 Pod 之间的关系，指定 Pod 之间的亲和性和反亲和性规则，用于将具有相似属性或标签的 Pod 调度到同一节点上或一组不相关的 pod 分散到不同的节点上。
- Node Affinity： 定义 Pod 和 Node 之间的关系，指定 Pod 对节点的偏好规则，用于将 Pod 调度到满足规则的节点上。

调度 Review

- 4种调度策略：NodeSelector、Node Affinity、Pod Affinity 和 Pod Anti-Affinity。
- 4种调度策略的使用场景

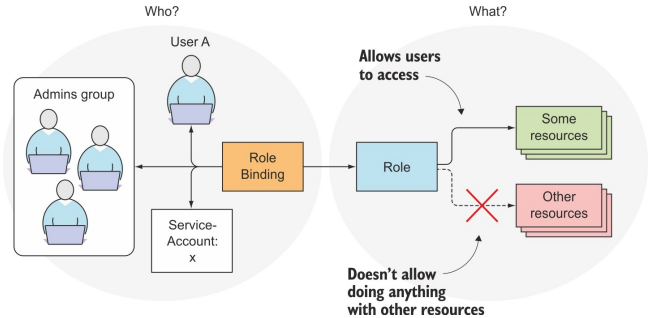
高可用



- 使用多个Master节点：Kubernetes集群通常由多个Master节点组成，其中一个Master节点作为主节点，其他节点作为备用节点。如果主节点发生故障，备用节点可以接管并继续提供服务。使用多个Master节点可以提高集群的冗余性和可用性
- 使用负载均衡器：将负载均衡器放置在Master节点和Worker节点之间，可以确保请求在多个节点之间进行均衡分配，从而提高集群的可用性。常用的负载均衡器包括Nginx、HAProxy等。
- 使用健康检查和自动恢复：Kubernetes提供了健康检查和自动恢复的功能，可以监测节点和容器的健康状态，并在发现故障时自动进行恢复。通过配置适当的健康检查和自动恢复策略，可以提高集群的可用性。
- 数据备份和恢复：定期对集群中的数据进行备份，并确保备份数据的可靠性和完整性。在发生故障时，可以使用备份数据进行快速恢复，减少服务中断时间。

安全

RBAC



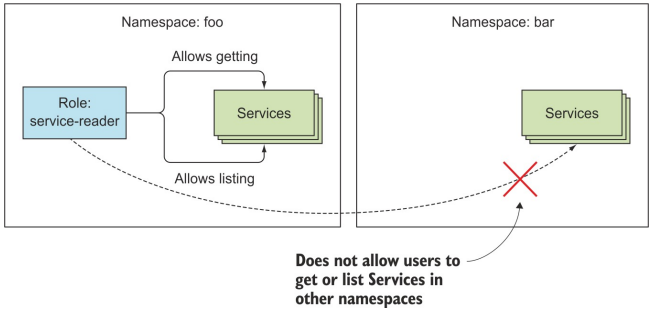
RBAC（Role–Based Access Control）是一种授权机制，用于管理对集群资源的访问权限。RBAC基于角色和权限的概念，通过定义角色和将角色分配给用户或用户组来控制对资源的访问。

- Role 和 ClusterRole，它们指定了在资源上可以执行哪些动作。
- RoleBinding 和 ClusterRoleBinding，它们将上述角色绑定到特定的用户、组或 ServiceAccount 上。

角色定义了可以做什么操作，而绑定定义了谁可以做这些操作。

安全

RBAC – Role

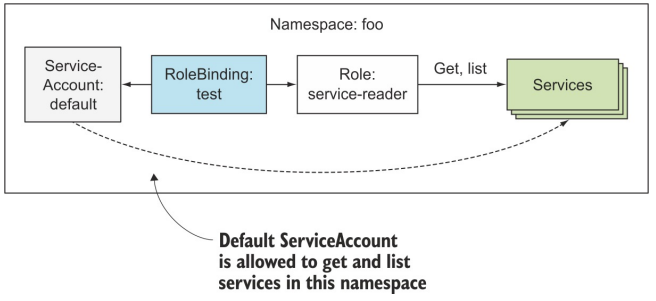


定义一个 Role，它允许用户获取并列出 foo 命名空间中的服务

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: foo
  name: service-reader
rules:
- apiGroups: [""]
  verbs: ["get", "list"]
  resources: ["services"]
```

安全

RBAC – RoleBinding

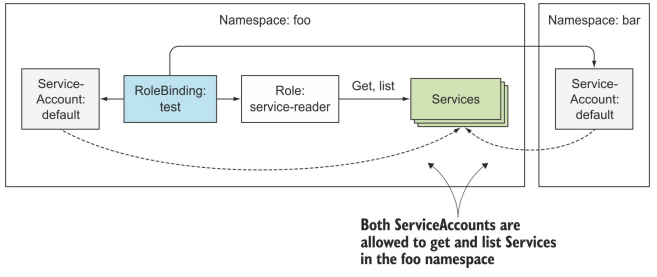


test RoleBinding 将 default ServiceAccount 和 service-reader Role 绑定

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test
  namespace: foo
  ...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: service-reader
subjects:
- kind: ServiceAccount
  name: default
  namespace: foo
```

安全

RBAC – RoleBinding



RoleBinding 将来自不同命名空间中的 ServiceAccount 绑定到同一个 Role

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: bar
```

安全

Network Policy

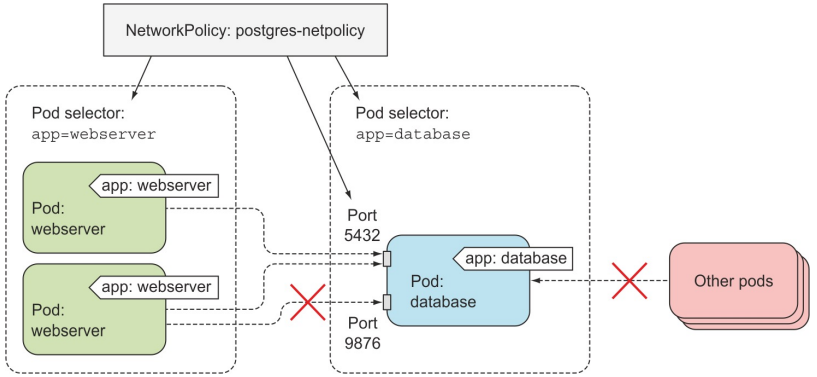
网络策略是一种用于控制Pod之间和Pod与外部通信的机制。它允许您定义规则，以确定哪些Pod可以与其他Pod进行通信，以及哪些Pod可以与集群外部的服务进行通信。

可以定义以下类型的规则：

- 入站规则：控制从其他Pod或集群外部进入Pod的流量。您可以指定允许或拒绝特定的IP地址范围、端口范围或协议。
- 出站规则：控制从Pod流出到其他Pod或集群外部的流量。您可以指定允许或拒绝特定的IP地址范围、端口范围或协议。
- 互联规则：控制Pod之间的通信。您可以指定允许或拒绝特定的标签选择器或命名空间。

安全

Network Policy – 同一命名空间

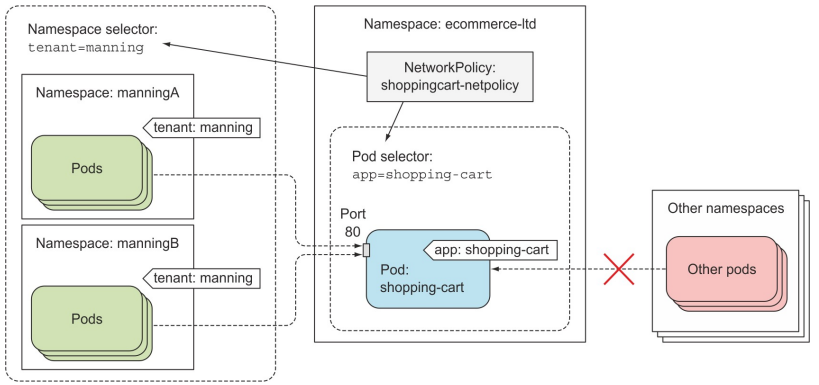


允许具有app=webserver标签的Pod访问具有app=database标签的Pod的NetworkPolicy，并且只能访问 5432 端口

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: postgres-netpolicy
spec:
  podSelector:
    matchLabels:
      app: database
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: webserver
    ports:
    - port: 5432
```

安全

Network Policy – 不同命名空间



仅允许匹配 namespaceSelector 的命名空间中的 pod 访问特定 pod 的NetworkPolicy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: shoppingcart-netpolicy
spec:
  podSelector:
    matchLabels:
      app: shopping-cart
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          tenant: manning
    ports:
    - port: 80
```

安全 Review

- 介绍了什么是 RBAC 以及如何使用
- 介绍了什么是 Network Policy 以及如何使用

Q & A