用例 4: 通过 Kubernetes 网络策略进行应用程序微分段

在大多数生产环境中，需要实施网络访问控制。Kubernetes 提供了一种方法来描述 Pod 组应该如何通过使用 NetworkPolicy 资源进行通信。

与 Kubernetes 中的大多数事情一样，要使网络策略正常运行，您需要一个支持它们的 Kubernetes CNI 插件。

使用场景

在几乎所有环境中，为应用程序需要通信的组件建立明确的规则，都是一个好主意。Kubernetes 网络策略规范是一种直接的方法，可让您将 `NetworkPolicy` 直接与应用程序清单集成在一起。
`NetworkPolicy` 定义资源的方式，使您可以精确地指定哪些网络通信是被允许的，而哪些则不允许，同时使用 `podSelector` 定义处理在 Kubernetes 上运行的应用程序的动态属性。

这意味着您的策略可以针对单个 Pod 或 Pod 组，从而将安全范围“缩小”到 Pod 的大小。

严格定义的网络策略与 `default-deny` 配置相结合，可以避免由于恶意应用程序入侵，和/或行为不当，或者配置错误而造成的麻烦。例如，一个应用程序组件可能具有滞留的缓存 DNS 条目或错误的配置参数，导致它与错误的后端进行通信。或者应用程序可能会被攻陷并被用作跳板，执行侦查，尝试横向渗透，或者只是使用 Pod 对 Kubernetes API 的访问权限来启动一些加密货币挖矿 Pod，以窃取您的计算资源。

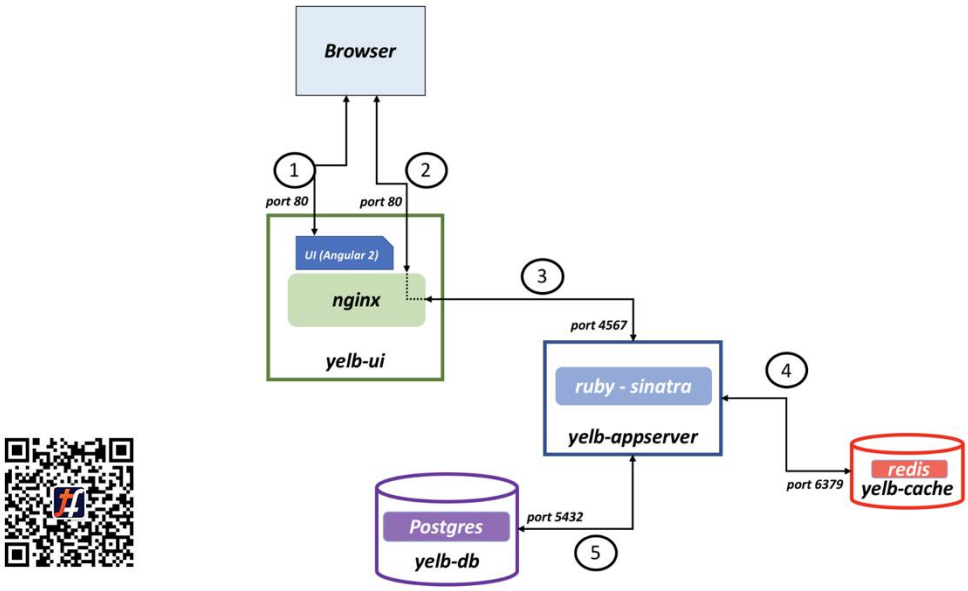
使用网络策略保护示例应用程序的安全

网络策略设计的主题比本指南中允许的空间要大得多。在此示例中，我们将执行以下操作：

- 为我们的 `default` 命名空间创建一个 `default-deny` Ingress 策略。这意味着命名空间内的所有传入 Pods 的连接必须明确被允许；
- 为每个示例应用程序组件创建一个 Ingress `NetworkPolicy` 对象，仅允许那些我们确定的对象。

步骤 1：明确哪些组件应当可以相互通信

首先，我们需要提醒自己，应用程序的各个组件应该如何通信。为此，我们将回到在简介中看到的应用程序图：



从该图中可以看到：

1. 外界需要到达 `yelb-ui` 的 TCP 端口 80-（1）和（2）
2. `yelb-ui` 需要到达 `yelb-appserver` 的 TCP 端口 4567
3. `yelb-appserver` 反过来将需要到达 `yelb-db` 的 TCP 端口 5432，以及
4. .. `yelb-cache` 的 TCP 端口 6379。

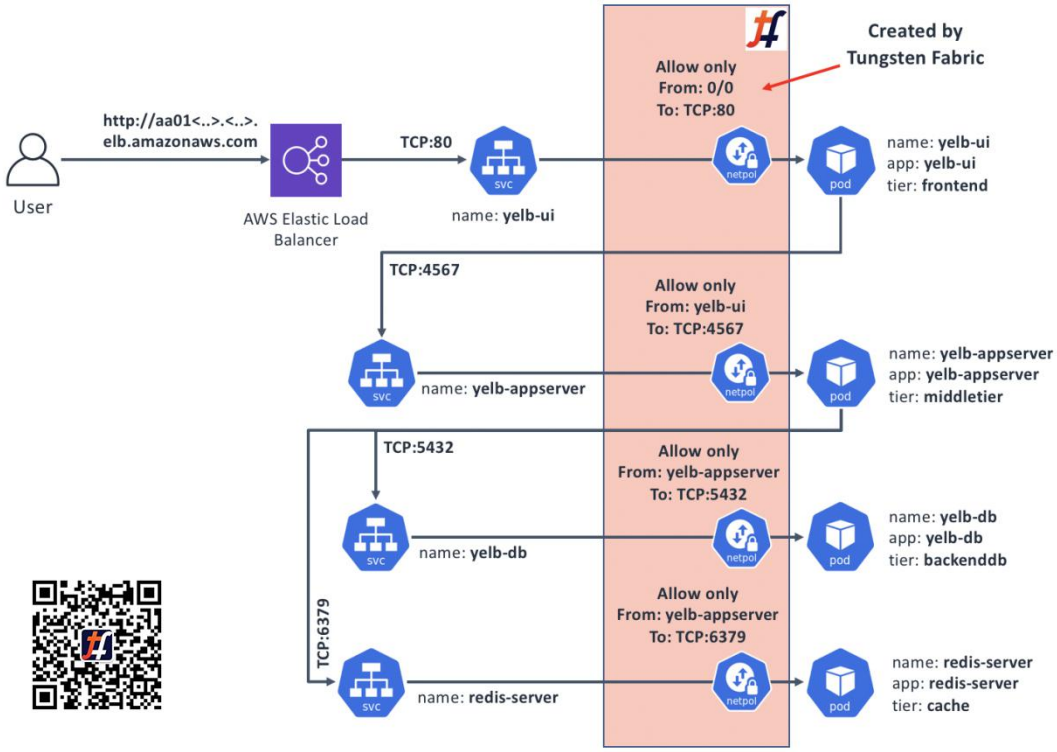
步骤 2：如何识别组件？

请记住，`NetworkPolicy` 资源使用选择器来识别策略适用于哪个 Pod，以及该策略将要控制的流量的源和目的地是什么。
在本演示中，我们将使用 `podSelectror` 方法，因此需要获取应用到应用程序 Pod 的标签列表。让我们查看 `cnawebapp-loadbalancer.yaml` 示例应用程序的清单，并收集标签：

Pod	标签
yelb-ui	应用：yelb-ui 层：frontend
yelb-appserver	应用：yelb-appserver 层：middletier
yelb-db	应用：yelb-db 层：backenddb
redis-server	应用程序：redis-server 层：cache

现在准备编写我们的策略。

部署后，这些策略将以以下方式控制应用程序组件之间的通信：



步骤 3: "default-deny"策略

确保您位于沙箱控制节点上，以 root 用户身份登录，并且位于正确的目录中：

```
# 确认您是 root 账户
whoami | grep root || sudo -s

# 切换到清单目录
cd
/home/centos/yelb/deployments/platformdeployment/Kubernetes/yaml
```

在此步骤中，我们将创建一个策略，该策略将阻止所有未明确允许的网络通信。在这一演示中，我们将只限制 `Ingress` 流量；但实际上，您也可以控制 `Egress` 流量（但是这样做时要注意这可能会阻止 DNS 查询！）：

```
cat > yelb-policy.yaml <<EOF
# First, add Ingress default-deny
#
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
```

```
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

该策略基本上说："对于任何 Pod，都应用一个没有规则的 `Ingress` 策略"，这将导致应用这个策略的，所有流向这个命名空间 Pods 的传入流量被丢弃掉。

步骤 4："yelb-ui"的策略

该 `yelb-ui` 和其他组件在某种意义上说有一些不同，因为它是唯一一个可以被从外部访问的组件。因此，其 `ingress:`定义将使用 `ipBlock` 的 `0.0.0.0/0`，以表示"每个人"：

```
cat >> yelb-policy.yaml <<EOF
---
# Policy to let anyone reach yelb-ui
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: yelb-ui-in
spec:
  podSelector:
    matchLabels:
      app: yelb-ui
      tier: frontend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 0.0.0.0/0
    ports:
    - protocol: TCP
      port: 80
EOF
```

该策略表示："对于具有应用标签 `app: yelb-ui` 和 `tier: frontend` 的 Pods，允许传入来自任何源 IP 的流量，只要它去往 Pod 的 TCP 端口 80"。

步骤 5：示例应用中其他 Pod 的策略

我们示例应用程序中的其他 3 个 Pod 仅会看到来自其他 Pod 的流量，因此其策略将使用带有允许发送流量的 Pod 标签的 `podSelector` 参数：

```
cat >> yelb-policy.yaml <<EOF
---
# Policy to let yelb-ui reach yelb-appserver
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: yelb-appserver-in
spec:
  podSelector:
    matchLabels:
      app: yelb-appserver
      tier: middletier
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: yelb-ui
          tier: frontend
    ports:
    - protocol: TCP
      port: 4567
---
# Policy to let yelb-appserver reach yelb-db
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: yelb-db-in
spec:
  podSelector:
    matchLabels:
        app: yelb-db
        tier: backenddb
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
```

```
      matchLabels:
        app: yelb-appserver
        tier: middletier
    ports:
    - protocol: TCP
      port: 5432
---
# Policy to let yelb-appserver reach redis-server
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: redis-server-in
spec:
  podSelector:
    matchLabels:
      app: redis-server
      tier: cache
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
      matchLabels:
        app: yelb-appserver
        tier: middletier
    ports:
    - protocol: TCP
      port: 6379
EOF
```

步骤 6：在应用策略之前测试

为了能有一个“前后”对比，让我们部署示例应用程序并获取基线：

```
# 部署我们的应用
kubectl create -f cnawebapp-loadbalancer.yaml
```

等待应用启动并在外部可用：

```
# 获得我们程序 yelb-ui Service 的外部 DNS 名字:
kubectl get svc -o wide | grep yelb-ui | awk '{print $4}'
```

我们应该可以看到类似 `a0b8dfc14916811e9b411026463a4a33-1258487840.us-west-1.elb.amazonaws.com` 的输出；在网络浏览器中打开它；样本应用程序应当加载了。

接下来，我们知道所有 Pod 通信都是不受限制的，因此我们应该能够从 `yelb-ui` ping 到 `yelb-db`——这是在应用程序正常运行且我们不进行故障排除动作的情况下，本来不应该发生的活动：

```
# 获得"yelb-ui"的完整 Pod 名字
src_pod=$(kubectl get pods | grep yelb-ui | awk '{print $1}')

# 获得"yelb-db"的 IP:
db_pod_ip=$(kubectl get pods -o wide | grep yelb-db | awk '{print $6}')

#  从"yelb-ui" ping"yelb-db":
kubectl exec -it ${src_pod} ping ${db_pod_ip}
```

我们应该看到该 `ping` 命令正在接收响应；因此存在不受限制的网络连接。按^ C 停止命令。

步骤 7：部署策略并测试结果

在最后一步，我们将部署策略并观察其效果：

```
# 部署网络策略
kubectl create -f yelb-policy.yaml
```

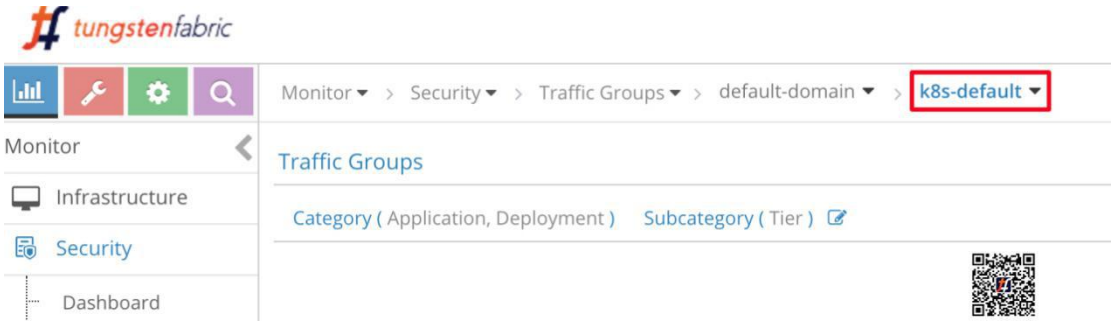
运行上面的命令后，请等待几秒钟以稳定下来。Tungsten Fabric 将在后台生成适当的安全组，并进行安装。让我们测试一下我们曾经可以正常运行的 ping 命令是否仍然有效：

```
# 从"yelb-ui" ping "yelb-db" again:
kubectl exec -it ${src_pod} ping ${db_pod_ip}
```

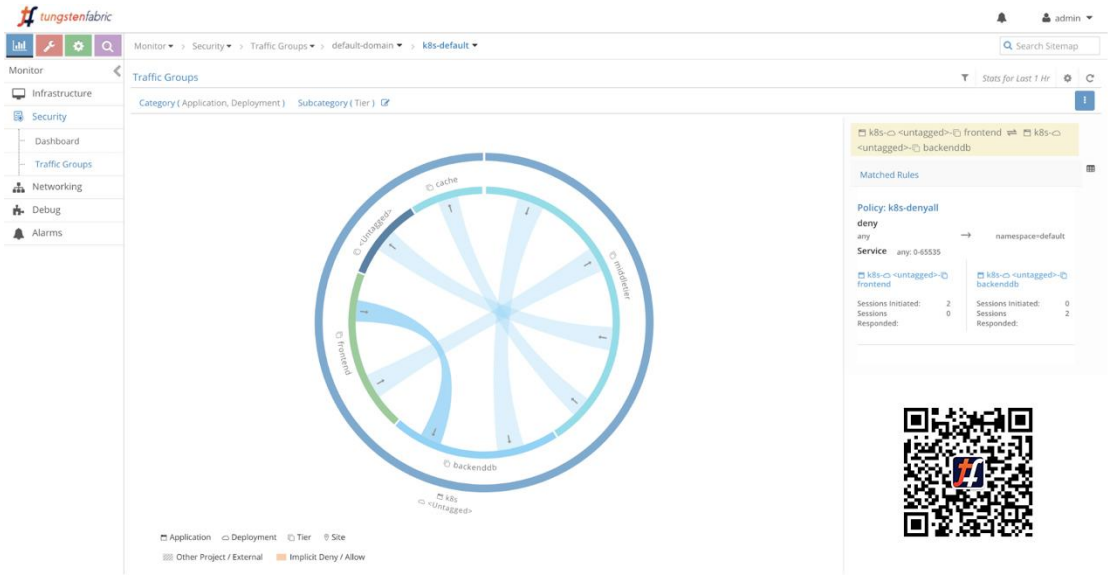
这次，我们看到没有响应，因为该通信现在已被该策略阻止。接下来，测试是否仍可以通过网络浏览器访问该应用——应该可以！

步骤 8：探索 Tungsten Fabric 的安全流量组可视化

Tungsten Fabric 包含一个功能，可在“项目”中实现流量可视化，在我们的案例中，该项目对应于 Kubernetes `Namespace`。
要访问它，请访问 `Carbide Evaluation Page` 链接，用于获取访问沙箱控制节点——在顶部有一个名为 `Contrail UI` 的链接，完成 `login` 和 `password` 的输入。单击链接，然后在左上角单击“Monitor”图标，然后在菜单中单击“Security” -> “Traffic Groups”。然后在顶部的标签链，在其末尾选择“k8s-default”：



您应该看到类似于以下的图表：



继续测试。您看到的流，代表示例应用程序在做的事情，包括无法从`yelb-ui`ping到`yelb-db`，以及`yelb-appserver`的出站请求（如果我们去查看，将转到`yelb-db`的 DNS 查询）。

清理

一旦进行了足够的探索，可以随时清理：

```
# 卸载 Network Policy
kubectl delete -f yelb-policy.yaml

# 删除我们的示例应用程序:
kubectl delete -f cnawebapp-loadbalancer.yaml

# 删除策略清单:
rm -f yelb-policy.yaml
```

回顾和下一步

对于许多（即使不是全部）生产部署，控制应用程序的网络通信能力至关重要。在 Kubernetes 上运行的应用程序实现此类控件的方法是通过 `NetwokPolicy` 资源。但是，要使这些资源真正起作用，您需要一个支持它们的 CNI 插件。Tungsten Fabric 提供了完整的 `NetworkPolicy` 支持，无论集成 Tungsten Fabric 的 Kubernetes 在哪里运行，是在私有数据中心，还是在公共云中。

网络策略可以变得非常简单或非常复杂，而找出最适合您的应用程序的最佳方法，就是在我们提供的用例和示例基础上更深入地研究。这里有一些资源可能会有所帮助：

- 网络策略的详细介绍
- 一个不错的 GitHub 存储库，其中包含大量具有详细文档的网络策略清单示例
- 在 2019 Kubecon 上的演讲中有一个介绍网络策略的很好的材料：
  https ://youtu.be/_VNv7jBh1XA?t=1273