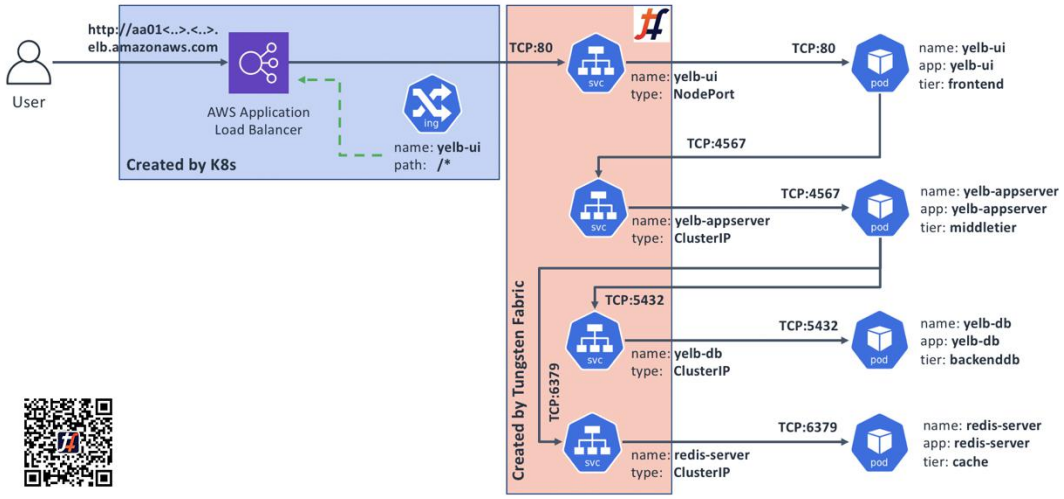
用例 2：通过 Kubernetes Ingress 进行高级外部应用程序连接

Kubernetes 的 Ingress 文档页面将其描述为：

“用于管理对集群中服务的外部访问的 API 对象，通常是 HTTP。Ingress 可以提供负载均衡、SSL 终结和基于名称的虚拟主机。”

CNI 不提供 Ingress 功能。这意味着 Kubernetes 集群管理者通常要为其集群安装、管理和支持单独的 Ingress 控制器解决方案。
对于没有内置 Ingress 支持的本地和公共云中的 Kubernetes 部署，Tungsten Fabric 捆绑了自己的 Ingress 控制器。它在后台使用 HAProxy 并实现了 Kubernetes Ingress 文档页面中所述的所有基本功能。
在 AWS 上运行时，可以将 Kubernetes 配置为使用 AWS 的 Application Load Balancer （ALB）为其 Ingress 服务。通过这种方式的设置，沙箱中的 Kubernetes 可以最紧密地反映典型的现实部署场景。

下图概述了示例应用程序的最终部署架构：



使用场景

Ingress 控制器选项仅与使用 HTTP 或 HTTPS 的应用程序兼容。如果您的应用程序是这种情况，可能需要考虑使用 Ingress 来实现以下目标：

- 使用 HTTPS 保护应用程序，然后通过配置 Ingress 进行 SSL 卸载来将程序公开在网络上；和/或
- 基于请求中的 HTTP 路径，将传入请求定向到不同的 Kubernetes Services，例如，`/blog/` 可以转到 `Service A`，而 `/account/` 可以转到 `Service B`，等等。和/或

- 通过基于名字的虚拟主机，应用程序服务于多个 DNS 域，例如 `Host:`头设置为 `test.project.com`的应用去 `Service C`，而那些具有 `prod.project.com`的去 `Service D`。

通过 Ingress 公开示例应用

在探讨上述三种情况之前，让我们部署一个简单的 Ingress 示例应用程序，类似于我们对 `LoadBalancer`的做法，然后在此基础上进行构建。

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

# 切换为清单目录
cd
/home/centos/yelb/deployments/platformdeployment/Kubernetes/yaml

# 部署具有 Ingress 的示例程序
kubectl create -f cnawebapp-ingress-alb.yaml
```

几分钟后，部署过程应该完成了，我们应该能够从 Internet 访问示例应用程序。首先找到 Ingress 的 DNS 名称：

```
# kubectl get ing yelb-ui
NAME      HOSTS   ADDRESS
PORTS   AGE
yelb-ui   *
539db10e-default-yelbui-3c9c-1330819777.us-west-1.elb.amazonaws.c
om   80      60m
```

根据上面的输出，现在可以从 Internet 上的 http://539db10e-default-yelbui-3c9c-1330819777.us-west-1.elb.amazonaws.com 上访问我们的示例应用程序。

利用在环境中运行上述命令获得的 DNS 名称访问 Yelb，以确保其有效。

使用 HTTPS 保护示例应用程序的安全

对于此练习，我们需要生成自签名证书，并将其添加到 AWS Certificate Manager。提供 Ingress 功能的 AWS Application Load Balancer（ALB）需要使用此功能来执行加密。

注意：对于生产用途，可能需要通过 AWS Certificate Manager 的相应功能来获得完整注册域名的“适当”证书。由于我们只是在进行练习，因此将使用自签名的虚构域。

步骤 1：生成自签名证书，并将其添加到 AWS Certificate Manager

在安装了具有 Access 和 Secret 密钥的 AWS CLI 工具的主机上执行以下步骤。这里的密钥允许您对 Certificate Manager 进行更改。

```
# 为虚构的域名（yelb.mydomain.com）生成自签名证书：
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key
-out tls.crt -subj "/CN=yelb.mydomain.com"

#将新的证书加入到 AWS Certificate Manager
# 注意“--region” -这必须是同一个 AWS 区域
# 在我们的例子中，运行 Tungsten 沙箱时，它是“us-west-1”
aws acm import-certificate --certificate file://tls.crt
--private-key file://tls.key --region us-west-1
```

如果一切顺利，最后一条命令将显示类似内容：

```
{
    "CertificateArn":
"arn:aws:acm:us-west-1:180612498884:certificate/e7341ff5-52ef-4a7
b-94b5-05643ef6ab46"
}
```

我们将需要 `CertificateArn` 后面的值，以进行下一步。

步骤 2：建立 Ingress 定义

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：

```
# 确定您是 root 身份
whoami | grep root || sudo -s

# 切换为清单目录
cd
/home/centos/yelb/deployments/platformdeployment/Kubernetes/yaml
```

现在，让我们创建一个新的 Ingress 定义：

```
cat > ingress-https.yaml <<EOF
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: "yelb-ui-https"
  annotations:
    kubernetes.io/ingress.class: alb
```

```
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'
    alb.ingress.kubernetes.io/certificate-arn:
INSERT_CERT_ARN_HERE
  labels:
    app: "yelb-ui"
spec:
  rules:
  - host: yelb.mydomain.com
    http:
      paths:
      - path: /*
        backend:
          serviceName: "yelb-ui"
          servicePort: 80
EOF
```

接下来，我们将 `CertificateArn` 放入。在运行该命令之前对其进行编辑，并用执行步骤 1 时获得 `arn:aws:acm:us-west-1:180612498884:certificate/e7341ff5-52ef-4a7b-94b5-05643ef6ab46` 的值替换该命令 `CertificateArn`。

```
sed -i
"s#INSERT_CERT_ARN_HERE#arn:aws:acm:us-west-1:180612498884:certif
icate/e7341ff5-52ef-4a7b-94b5-05643ef6ab46#" ingress-https.yaml
```

如果命令成功运行，则 `ingress-https.yaml` 文件将具有自签名证书的 ARN，而不是字符串 `INSERT_CERT_ARN_HERE`。

步骤 3：建立 HTTPS Ingress 并进行测试

```
# 创建新的 Ingress
kubectl create -f ingress-https.yaml
```

运行上述命令后，请等待几分钟，以使新的 ALB Ingress 出现。然后，让我们找到已为其分配的 DNS 名称，并尝试连接到它：

```
# kubectl get ingress yelb-ui-https
NAME           HOSTS              ADDRESS
PORTS   AGE
yelb-ui-https   yelb.mydomain.com
539db10e-default-yelbuihtt-2a0d-1983111448.us-west-1.elb.amazonaw
s.com   80      16m
```

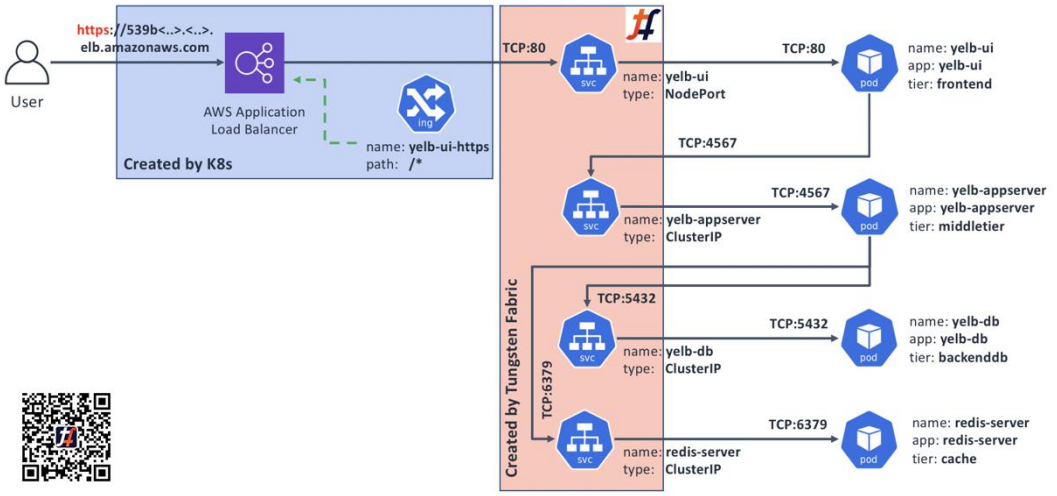
从上面的输出中，我们可以看到新 Ingress 的地址；让我们看看它是否有效：

```
# curl -v -k
https://539db10e-default-yelbuihtt-2a0d-1983111448.us-west-1.elb.
amazonaws.com -H "Host: yelb.mydomain.com"
```

```
[..skip..]

* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*       subject: CN=yelb.mydomain.com

[..skip..]

<html>
<head>
    <script src="env.js"></script>
    <meta charset="utf-8">
    <title>Yelb</title>
    <base href="/">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico?v=2">
</head>
<body>
<yelb>Loading...</yelb>

[..skip..]
```

这说明它有效——我们可以通过加密连接访问 Yelb 应用程序！

新拓扑看起来像这样（请注意，我们仍然具有未在此图中显示的原始 HTTP Ingress）：



总结：为什么要这样做?

除了增加最终用户的连接安全性、隐私性和数据完整性外，实现 HTTPS Ingress 还有一些好处：

- 应用程序消耗较少的计算资源，因为加密开销已转移到 ALB。
- 应用程序现在支持 HTTP / 2，这是一件好事;
- 可以轻松实现将 HTTP 自动重定向到 HTTPS 的功能。

清理

让我们删除添加的 HTTPS Ingress，因为在本章的其余部分中我们不再需要它：

```
kubectl delete -f ingress-https.yaml
```

然后，在执行步骤 1（生成自签名证书并将其安装到 AWS Certificate Manager 中）的计算机上，运行以下命令以删除该证书，并确保使用您自己的值 `CertificateArn`：

```
aws acm delete-certificate --certificate-arn
arn:aws:acm:us-west-1:180612498884:certificate/e7341ff5-52ef-4a7b
-94b5-05643ef6ab46
```

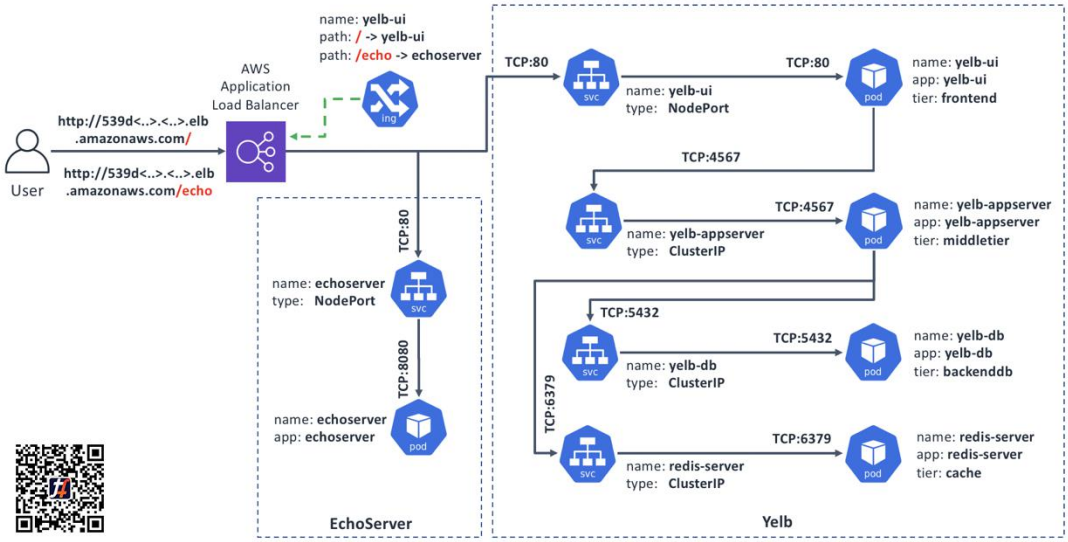
根据 URL 路径定向请求

在某些情况下，您可能想在同一个 DNS 域名下运行多个应用程序。例如，`www.corp.com` 可能支持您的主应用程序，而诸如 WordPress 之类的其他应用程序可能正在处理 `www.corp.com/blog`。

对于本练习，我们假设您在“通过 Ingress 公开示例应用程序”这一章的开头按照说明运行了 Yelb 副本。如果您是从头开始，请跳至该部分，按照说明进行部署，然后再回来。

为了演示通过 URL 路径进行的路由，我们将在环境中添加另一个部署，并相应地更新 Ingress 的配置。在此新配置下，Ingress 会将对 `/` 路径的请求定向到我们的主应用程序 `yelb`，而对 `/echo` 的路径请求将定向为新的应用程序 `EchoServer`。

这是目标状态的图：



我们应该已经将 `Yelb` 的部分放置到位，所以我们添加 `Echoserver`：

```
# 创建 EchoServer Deployment and Service 清单:
cat > echoserver.yaml << EOF
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: echoserver
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: echoserver
    spec:
      containers:
      - image: gcr.io/google_containers/echoserver:1.4
        name: echoserver
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
```

```
  type: NodePort
  selector:
    app: echoserver
EOF

# 现在部署它:
kubectl create -f echoserver.yaml
```

接下来，我们将为 Ingress 创建一个更新的配置。为此，我们将从中复制 Ingress 资源 `cnawebapp-ingress-alb.yaml`，并在路由部分进行两项更改：

1. 将 `yelb` 的路径从 `/*` 更新到 `/` 以免干扰 `echoserver`；和
2. 添加新的 `/echo` 路径指向 `echoserver`

注意：我们之所以要包含完整的资源定义而不是仅仅应用差异部分，是因为 `Ingress` 对象不支持战略性合并修补。

```
# 更新的 Ingress 资源:
cat > ingress-paths.yaml << EOF
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: "yelb-ui"
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
  labels:
    app: "yelb-ui"
spec:
  rules:
    - http:
        paths:
          - path: /
            backend:
              serviceName: "yelb-ui"
              servicePort: 80
          - path: /echo
            backend:
              serviceName: "echoserver"
              servicePort: 80
EOF

# 现在部署它:
kubectl apply -f ingress-paths.yaml
```

这里会显示一个关于 `kubectl apply` 的警告，这个警告可以忽略。因为我们的更新资源在本质上与 `rules` 配置相同。

更新的配置在几秒钟内生效，之后我们就可以检查基于 URL 的路由是否有效。当我们请求基本 URL `/`（或为空）时，我们应该到达 `yelb`，如果请求`/echo`，我们应该返回的输出是 `EchoServer`。

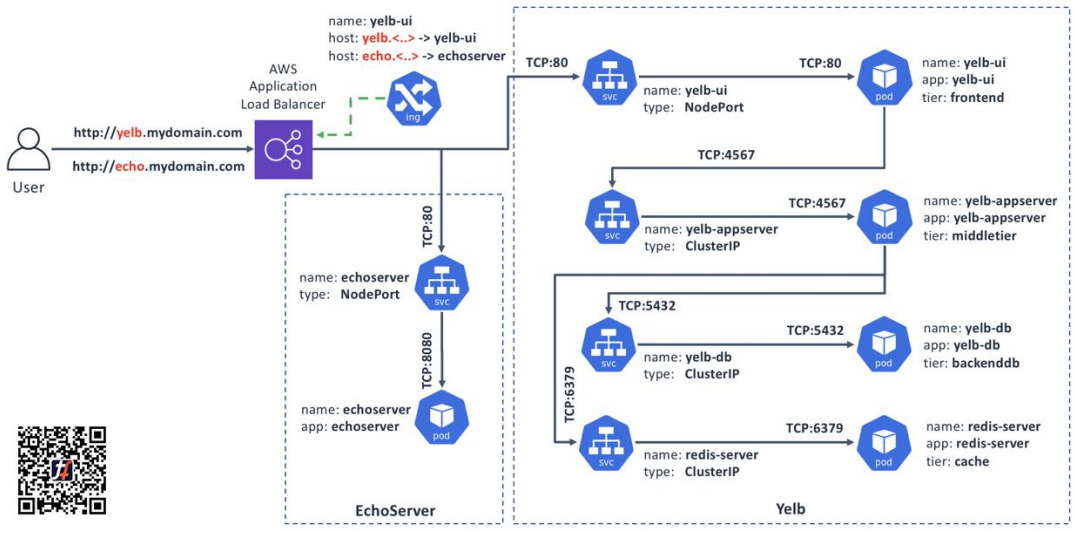
```
# 获得 Ingress 的基本 URL
baseUrl=$(kubectl get ing yelb-ui | grep amaz | awk '{print $3}')
echo "Our Ingress is at: ${baseUrl}"

# 尝试访问$baseUrl；应当可以得到 Yelb UI 页面的内容
curl http://${baseUrl}

# 现在尝试/echo；应当得到 EchoServer 的输出
curl http://${baseUrl}/echo
```

服务多个 DNS 域

当您拥有多个域名，并且为每个域提供不同的应用程序，同时希望共享相同的 Ingress 基础结构，此场景中的解决方案就很有用。这有助于节省成本，并且在某些情况下，与每个域名拥有专用的 Ingress 实例相比，其复杂性更低。

此练习建立在上一个基于 `URL 定向请求`的练习的基础上。如果尚未完成，请回顾此前步骤，简单地剪切并粘贴创建和部署 `echoserver.yaml` 清单的步骤。我们将为 Ingress 新建一个，因此无需创建和部署 `ingress-paths.yaml`。
准备好后，您应该已经有了 `yelb` 的副本和 `echoserver` 的副本。您的 Ingress 配置是什么都无所谓，因为我们将覆盖它。
在我们的目标状态下，Ingress 将定义两个域名 `yelb.mydomain.com` 和 `echo.mydomain.com`，并将根据 `Host:`HTTP 头中的值来路由传入的请求，Web 浏览器会自动为您请求的 URL 的主机部分插入这些请求。

这是目标状态的图：



让我们为 Ingress 创建并部署配置，该配置将执行所需的路由：

```
# 更新的 Ingress 资源:
cat > ingress-hosts.yaml << EOF
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: "yelb-ui"
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
  labels:
    app: "yelb-ui"
spec:
  rules:
    - host: yelb.mydomain.com
      http:
        paths:
          - path: /*
            backend:
              serviceName: "yelb-ui"
              servicePort: 80
    - host: echo.mydomain.com
      http:
        paths:
          - path: /*
            backend:
              serviceName: "echoserver"
              servicePort: 80
EOF

# 现在部署它:
kubectl apply -f ingress-hosts.yaml
```

配置成功应用后，我们就可以进行测试了。由于域名和主机形成映射，因此我们将利用 `curl` 添加正确的 `Host:` 标头。当设置为 `yelb.mydomain.com`，应该到达 `Yelb`，设置为 `echo.mydomain.com` 时，应该返回输出 `EchoServer`。

```
# Get the base URL of our Ingress
baseUrl=$(kubectl get ing yelb-ui | grep amaz | awk '{print $3}')
echo "Our Ingress is at: ${baseUrl}"

# 访问 Ingress Host: 设置成 yelb; 我们应当可以得到 Yelb UI 页面的内容
curl http://${baseUrl} -H "Host: yelb.mydomain.com"

# 现场尝试访问 Host:设置成 echo;我们应当可以得到 EchoServer 的输出
curl http://${baseUrl} -H "Host: echo.mydomain.com"
```

清理

一旦进行了足够的测试，请随时清理：

```
# 删除“yelb”和“echoserver”应用:
kubectl delete -f cnawebapp-ingress-alb.yaml
kubectl delete -f echoserver.yaml

# 删除我们创建的额外的清单:
rm -f echoserver.yaml ingress-paths.yaml ingress-hosts.yaml
```

回顾和下一步

Kubernetes 提供了三种将应用程序公开给外界的基本方式： `LoadBalancer` 或 `NodePort` 服务类型以及 Ingress。前两个协议支持任意协议，但在协议智能方面并没有增加太多。另一方面，Ingress 提供了基于协议的功能，这使其仅与 HTTP 或 HTTPS 的应用程序兼容。
与其他功能类似，Kubernetes 需要一个控制器来实现实际的 Ingress 功能——简单地在 Kubernetes API 中创建 `Ingress` 资源并不能执行任何操作。Ingress 控制器是 Kubernetes 集群管理员必须安装、监视、修补和升级的软件的一部分。Tungsten Fabric 随附有 Ingress 控制器，这将使此过程更加容易。

一旦确定了应用程序该如何公开于 Internet，就需要考虑如何处理有关网络访问控制的问题。阅读本指南中的用例 3 和用例 4，我们将介绍其中的一些场景。