

Nginx Plus Ingress Controller Workshop

Welcome

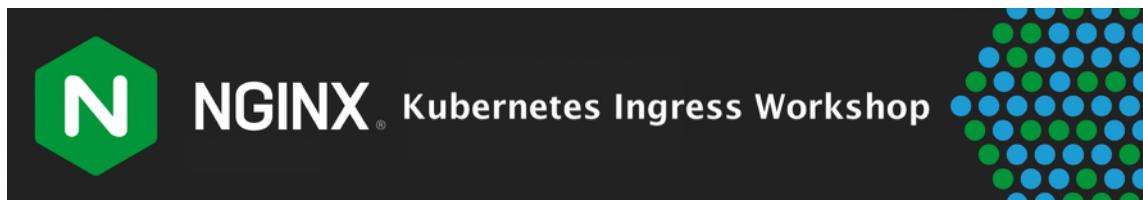
Welcome to the Nginx Plus Workshop for Kubernetes Ingress Controller (KIC)! This Workshop will introduce Nginx Plus Ingress Controller with hands-on practice through self-paced lab exercises.

You will learn how to configure an **Nginx Plus Ingress Controller**, deploy it on a Kubernetes cluster, configure advanced Nginx Plus features, loadtest it, scale it up and down and monitor it in realtime. You will deploy new apps and services in your private cluster, terminate SSL, route HTTP traffic, configure redirects, set up healthchecks, and load balance traffic to running pods.

These Hands-On Lab Exercises are designed to build upon each other, adding additional services and features as you progress through them, completing the labs in sequential order is required.



By the end of this Workshop, you will have a working, operational Nginx Plus Ingress Controller, routing traffic to and from Kubernetes application pods and services, with the necessary skills to



Nginx Plus Ingress Controller Workshop

Overview

Welcome to the Nginx Plus Workshop for Kubernetes Ingress Controller (KIC)!

This Workshop will introduce Nginx Plus Ingress Controller with hands-on practice through self-paced lab exercises. You will learn how to deploy an Nginx Plus Ingress Controller on a Kubernetes cluster, configure basic and advanced Nginx Plus features; and then test it, and monitor it in realtime.

Nginx Plus Ingress	Hands-On Labs
A green hexagonal icon containing a white stylized 'N'. Three green arrows point towards the 'N' from the left side. A green octagonal icon containing a white steering wheel.	An illustration of a person wearing a green hoodie and blue jeans, sitting cross-legged on a green cube, working on a laptop.

The Hands-On Lab Exercises are designed to build upon each other, adding additional services and features as you progress through them. It is important to complete the lab exercises in sequential order.

By the end of this Workshop, you will have a working, operational Nginx Plus Ingress Controller, with the skills to deploy and operate one for your Digital Enterprise Kubernetes projects.

Prerequisites

- Students: UDF Account Registered with an Email Address

Client Machine

- Local Computer with Microsoft Remote Desktop application installed.

Lab Client Machine

- Ubuntu Jumphost
- Chrome Browser
- All labs are completed using RDP to an Ubuntu Jumphost.

Ubuntu	Chrome	Remote Desktop
		

**** Prerequisite Knowledge ****

It is highly recommended for Students attending this Workshop to be familiar with Nginx and have some experience with Kubernetes administration, networking tools, and Load Balancing concepts. Previous experience with VisualStudio Code is also helpful.

VisualStudio Code	Kubernetes	Nginx Plus
		

An excellent Prerequisite for taking this Workshop is the Nginx Basics Workshop, also available from F5 Networks, as it provides a fundamental basics of Nginx Plus, on which the Nginx Plus Ingress Controller is built. Please contact your Nginx Event Marketing team for information on additional Workshops.

Lab Outline

Lab 1: Lab Overview and Student access

- [Lab 1: Workshop Lab Overview](#)

Lab 2: Verify Nginx Plus Ingress Controller is running

- [Lab 2: Verify Nginx Plus Ingress Controller is running](#)

Lab 3: Configuring Nginx Ingress Controller

- [Lab 3: Configure Nginx Ingress](#)

Lab 4: Access Nginx Plus Dashboard

- [Lab 4: Enable Nginx Plus Dashboard](#)

Lab 5: Deploy Nginx Cafe Demo using manifests

- [Lab 5: Deploy Nginx Cafe Demo](#)

Lab 6: HTTP load testing the Cafe Application

- [Lab 6: HTTP Load Testing](#)

Lab 7: Scaling up/down the Cafe Application

- [Lab 7: Scaling Cafe Ingress](#)

Lab 8: Monitoring KIC with Prometheus and Grafana

- [Lab 8: Prometheus and Grafana](#)

Lab 9: Retail App using Virtual Server / VSRoute

- [Lab9: VirtualServer/VSRoute](#)

Lab 10: Advanced Nginx Plus features with VirtualServer/VSRoute

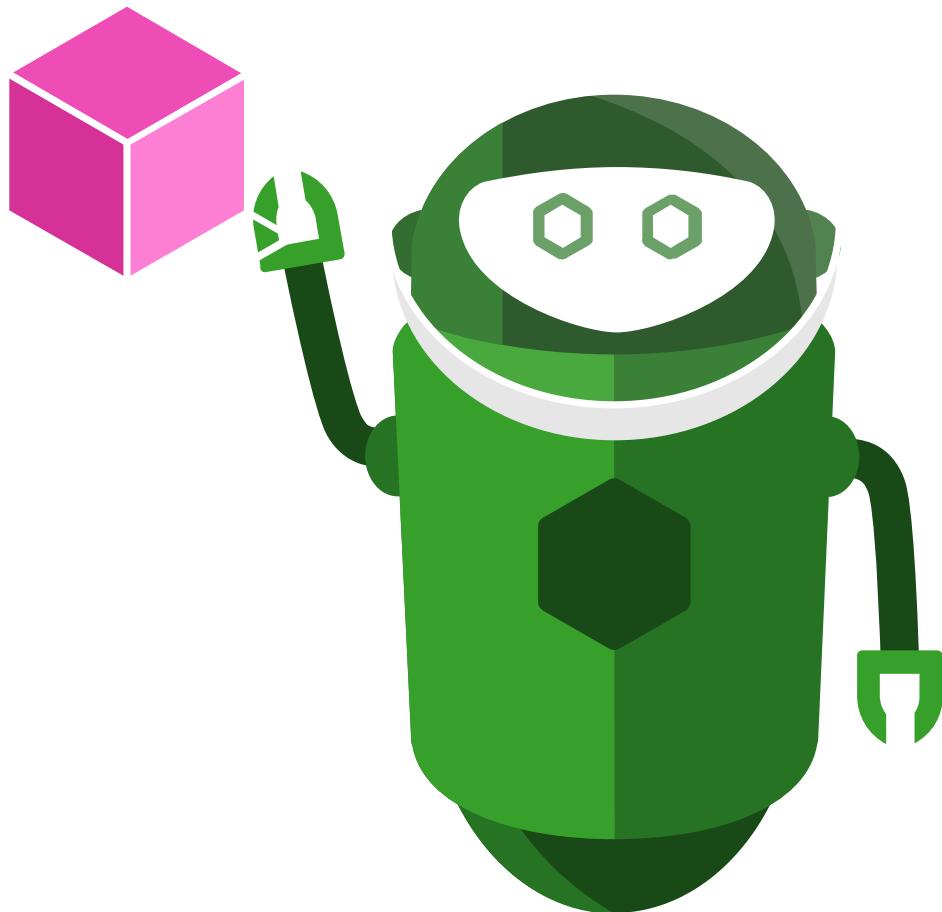
- [Lab10: Advanced Nginx Plus](#)

Authors

- Jason Williams - Product Management Engineer @ F5, Inc.
- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

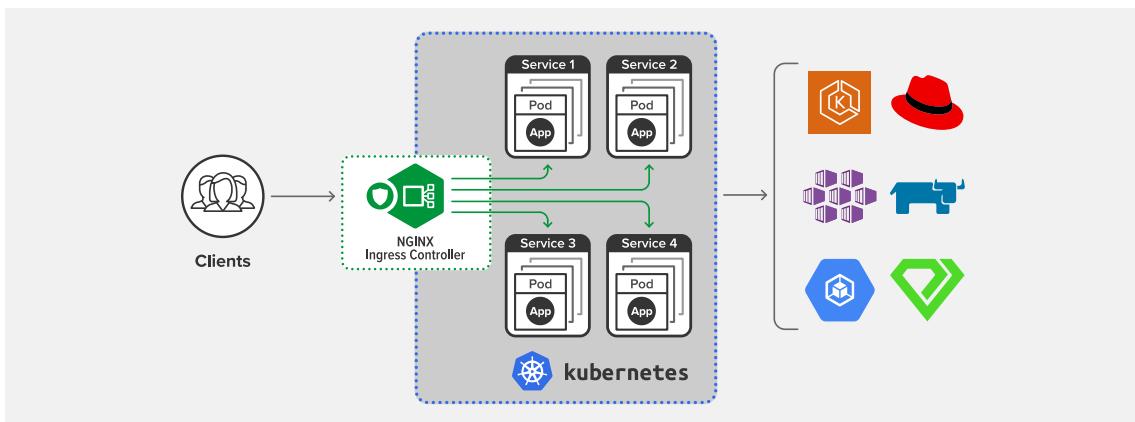
Click [Lab1: Workshop Lab Overview](#) to get started!

deploy and operate KIC for your own Digital Enterprise Applications running in Kubernetes. Thank You for taking the time to attend this Nginx Workshop!



About Nginx Plus Ingress Controller

Nginx Plus Ingress Controller is an Enterprise-Ready resource for directing traffic to/from a Kubernetes Cluster. Nginx runs all popular K8s frameworks, including Amazon EKS, Google GCP, Redhat Openshift, Azure AKS and others. It can also work with your on-premise Data Center based K8s clusters. Nginx Plus Ingress Controller is built from the same source code you know and trust from Nginx Plus. You can find the full Kubernetes support matrix and technical specifications for Nginx Ingress Controller on the <http://www.nginx.com> website.



Nginx Ingress Controller has the best-in-class traffic management solution for cloud-native apps in Kubernetes and containerized environments. In a recent [CNCF survey](#), nearly two-thirds of respondents reported using the Nginx Ingress Controller, more than all other controllers combined – and Nginx Ingress Controller has been downloaded more than [10 million times](#) on DockerHub.

NGINX+

Combining the speed and performance of Nginx with the trust and security behind F5 Networks, Nginx Plus Ingress Controller is synonymous with high-performing, scalable, and secure modern apps in production.

<https://docs.nginx.com/nginx-ingress-controller/intro/nginx-plus/>

<https://docs.nginx.com/nginx-ingress-controller/technical-specifications/>

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.
- Jason Williams - Product Management Engineer @ F5, Inc.

This completes the Introduction.

Click on ([LabGuide](#)) to begin the workshop.

Lab 1: Lab Overview and Student Access

Introduction

In this lab you will setup the workshop environment and prepare the Jumphost where you will be doing the labs from. You will be using the Microsoft RDP protocol to connect to an Ubuntu Desktop Jumphost. It has VisualStudio Code editor with a built-in bash terminal with tools like `kubectl` and `curl` already loaded to be used by the student.

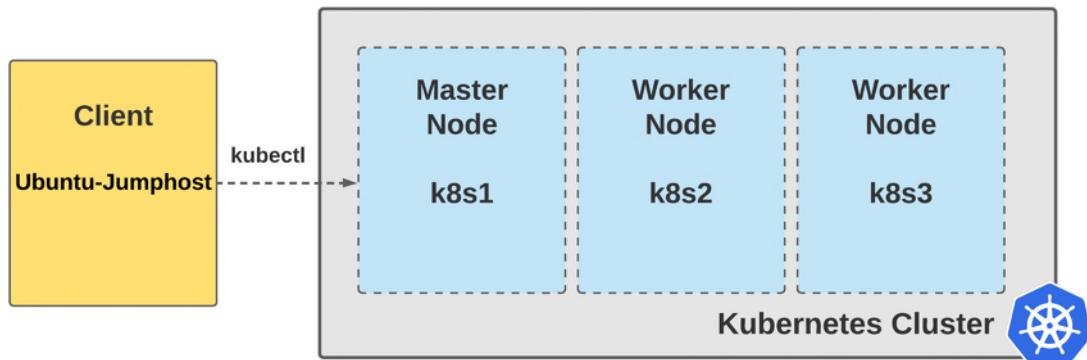
Important: All lab exercises must be run from the Ubuntu Desktop Jumphost in order to complete them successfully.

Learning Objectives

By the end of the lab, you will be able to:

- Understand the components of the Workshop environment
- Connect to Jumphost with an RDP(Remote Desktop Protocol) client
- Ready to start the Workshop labs

The Basic Architecture of the lab is shown here for reference:



Access Jumphost

Prepare the your client machine for this lab by:

- Opening the LabGuide
- Opening NginxPlus KIC Workshop vs-code workspace shortcut, located on the desktop of the Jumphost:

Locate the `Ubuntu-Jumphost` component and click on the `ACCESS` dropdown link.

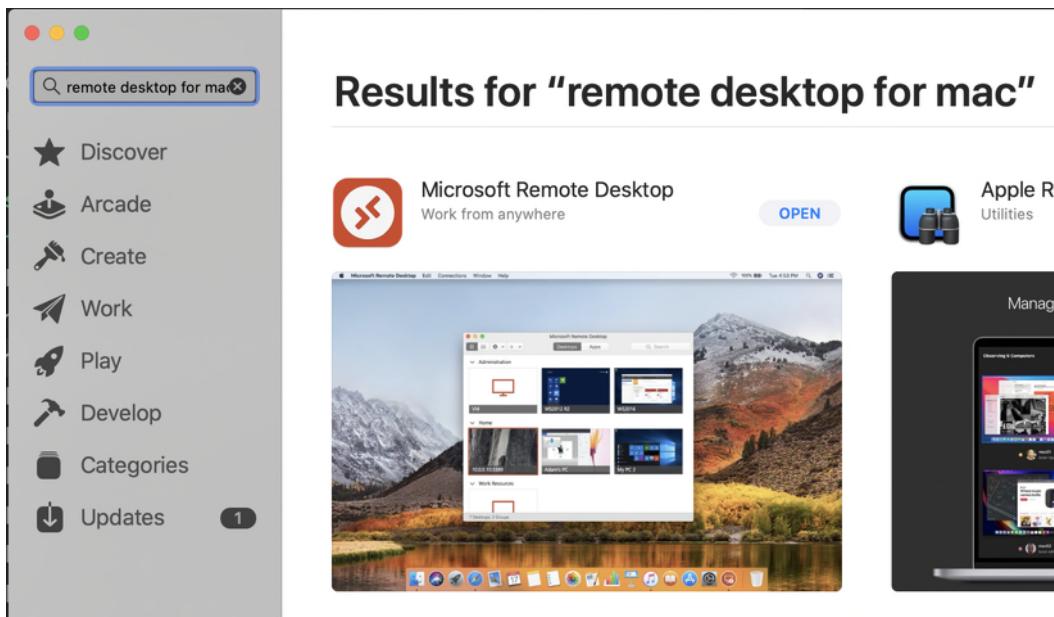
Click WORKSHOP LAB GUIDE , which should open the Lab Guide in a seperate tab.

The screenshot shows a web browser window titled "NGINX Plus Kubernetes Ingress Controller Workshop". The URL is "localhost:9999/LabGuide.md". The page features a large green hexagonal NGINX logo on the left and the text "NGINX® Kubernetes Ingress Workshop" next to it. To the right is a decorative graphic of colored dots in blue, green, and yellow. Below the header, the title "Nginx Plus Ingress Controller Workshop" is displayed. Underneath, there's a section titled "Overview" with a brief introduction and a "Hands-On Labs" section containing two icons: one showing a server with an arrow pointing to it, and another showing a person sitting at a desk with a laptop.

The screenshot shows the "Lab Guide" page from the workshop. The URL is "localhost:9999/LabGuide.md". The page has a header "NGINX Plus Kubernetes Ingress Controller Workshop". Below the header, there's a section titled "Lab Outline" which lists ten labs with their descriptions and sub-tasks:

- Lab 1: Lab Overview and Student access
 - Lab 1: Workshop Lab Overview
- Lab 2: Verify Nginx Plus Ingress Controller is running
 - Lab 2: Verify Nginx Plus Ingress Controller is running
- Lab 3: Configuring Nginx Ingress Controller
 - Lab 3: Configure Nginx Ingress
- Lab 4: Access Nginx Plus Dashboard
 - Lab 4: Enable Nginx Plus Dashboard
- Lab 5: Deploy Nginx Cafe Demo using manifests
 - Lab 5: Deploy Nginx Cafe Demo
- Lab 6: HTTP load testing the Cafe Application
 - Lab 6: HTTP Load Testing
- Lab 7: Scaling up/down the Cafe Application
 - Lab 7: Scaling Cafe Ingress
- Lab 8: Monitoring KIC with Prometheus and Grafana
 - Lab 8: Prometheus and Grafana
- Lab 9: Retail App using Virtual Server / VSRoute
 - Lab9: VirtualServer/VSRoute
- Lab 10: Advanced Nginx Plus features with VirtualServer/VSRoute
 - Lab10: Advanced Nginx Plus

1. To access this Workshop, and complete the lab exercises, you will need a Remote Desktop client software installed on your system. Windows PCs should already have Microsoft's Remote Desktop Client software installed. Mac users may need to install Microsoft RDP Client from the Apple App Store (free). <https://apps.apple.com/us/app/microsoft-remote-desktop/id1295203466?mt=12>



2. Under ACCESS now click on XRDП to login to the Jumphost

A screenshot of the UDF Access page. It displays two components: "Ubuntu-Jumphost" (Ubuntu 20.04 LTS Desktop) and "k8s3" (Ubuntu 18.04 LTS Server). Both are listed as "Running". The "ACCESS" dropdown menu is open for both components, showing options: CONSOLE, XRDП (highlighted with an orange circle), SSH (47004), WORKSHOP LAB GUIDE (highlighted with an orange circle), and WEB SHELL. The "DETAILS" tab is also visible for each component.

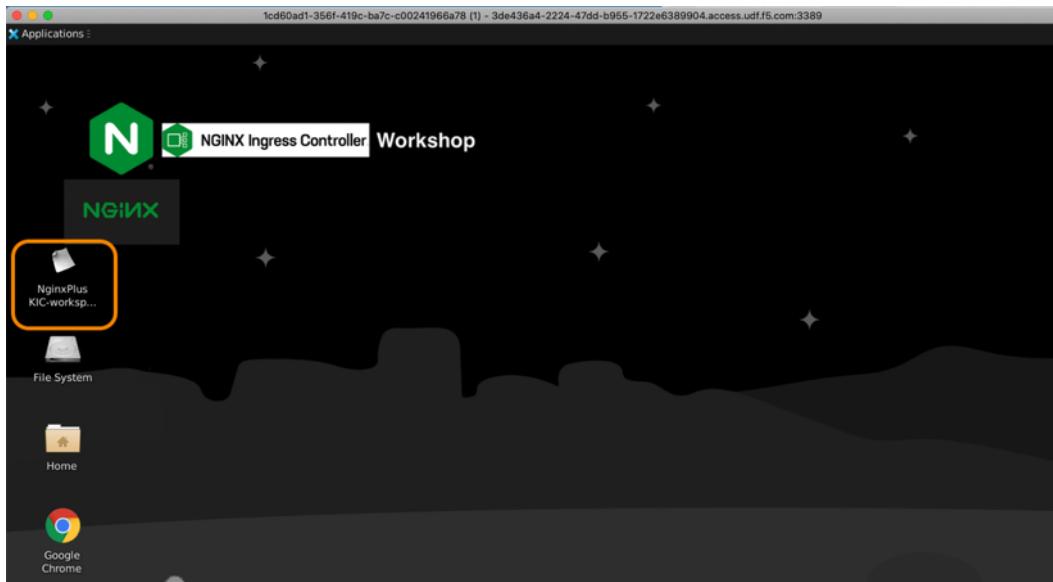
Note that you can re-open these components from the UDF Access page at any time.

The XRDП session will use the below login credentials:

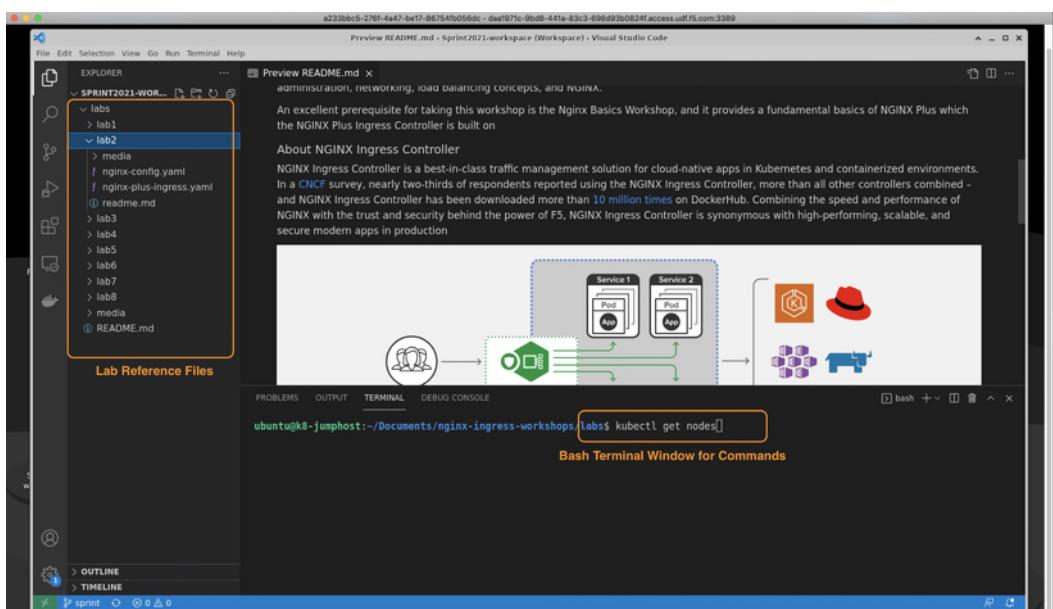
- username: **ubuntu**
- password: **Nginx123**



3. After you have logged into the Jumphost, open NginxPlus KIC Workshop Workspace to open VScode in the workshop project directory. You will be running all the lab exercises in VS Code and its built-in Terminal to run commands like `kubectl`, `curl`, `docker` and much more.



4. Using the VScode Terminal (the bottom pane), go ahead a try a `kubectl` command, such as `kubectl get nodes`.

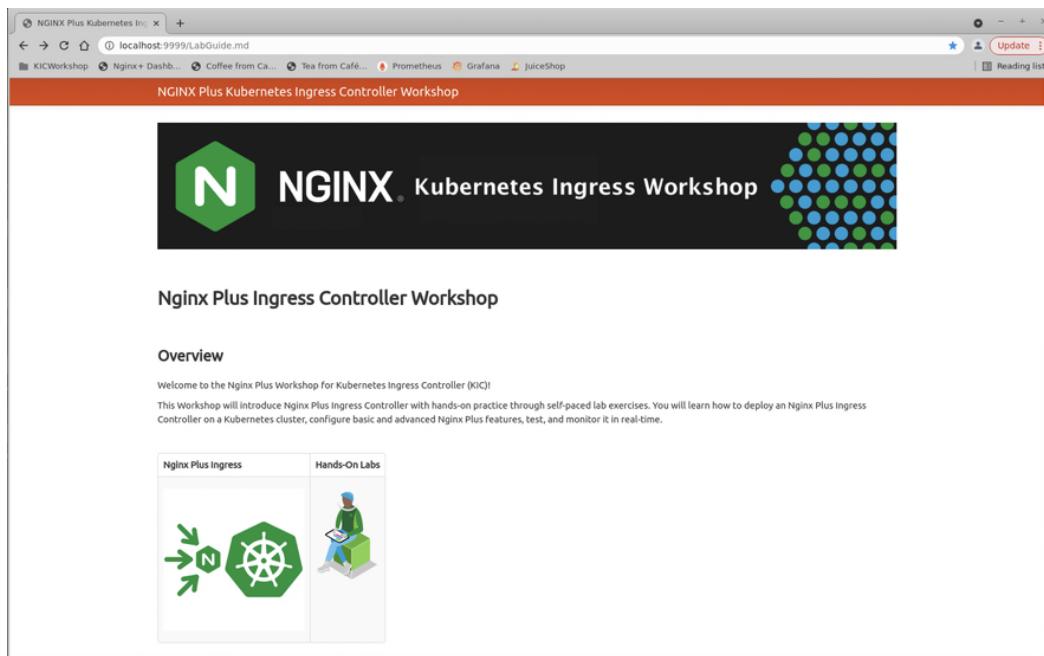


Verify your Kubernetes Cluster is up and all Nodes are in the "Ready" state using this command:

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get nodes
NAME    STATUS    ROLES      AGE     VERSION
k8s1   Ready     master     340d    v1.21.2
k8s2   Ready     <none>    340d    v1.21.1
k8s3   Ready     <none>    340d    v1.21.1
```

NOTE: If your nodes are not showing in `Ready` status then please inform your instructor.

5. As part of the lab, you will be using `Chrome` browser which you can also find on the Jumphost desktop, which also has the Lab Guide and some other links bookmarked which will be used for subsequent labs.



This completes this Lab.

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab2](#) | [Main Menu](#))

Lab 2: Verify Nginx Plus Ingress Controller is running

Introduction

The Nginx Ingress Controller is already running in this workshop. You will be checking and verifying the Ingress Controller is running.

Learning Objectives

- Intro to Nginx Ingress Controller
- Intro to Kubernetes environment, interacting with `kubectl` command
- Access the NginxPlus Dashboard

Check your Ingress Controller

1. First, verify the Nginx Ingress controller is up and running correctly in the Kubernetes cluster:

```
kubectl get pods -n nginx-ingress
```

```
# Should look similar to this...
NAME                  READY   STATUS    RESTARTS   AGE
nginx-ingress-fd4b9f484-t5pb6   1/1     Running   1          12h
```

2. Instead of remembering the unique pod name, `nginx-ingress-xxxxxx-xxxx`, we can store the Ingress Controller pod name into the `$KIC_POD_NAME` variable to be used throughout the lab.

Note: This variable is stored for the duration of the terminal session, and so if you close the terminal it will be lost. At any time you can refer back to this step to save the `$KIC_POD_NAME` variable once again

Note: You must use the `kubectl " -n "`, namespace switch, followed by namespace name, to see pods that are not in the default namespace.

```
export KIC_POD_NAME=$(kubectl get pods -n nginx-ingress -o
jsonpath='{.items[0].metadata.name}')
```

Verify the variable is set correctly.

```
echo $KIC_POD_NAME
```

Note: If this command doesn't show the name of the pod then run the previous command again.

Inspect the details of your Ingress Controller:

1. Inspect the details of the NGINX Ingress Controller pod using the `kubectl describe` command

```
kubectl describe pod $KIC_POD_NAME -n nginx-ingress
```

Note: The IP address and TCP ports that are open on the Ingress (they should match the `lab2/nginx-plus-ingress.yaml` file, around lines 24-34). We have the following Ports:

- Port 80 and 443 for web/ssl traffic,
- Port 8081 for Readiness,
- Port 9000 for the Dashboard, and
- Port 9113 for Prometheus (You will see this in a later Lab)

```
~Ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl describe pod $KIC_POD_NAME -n nginx-ingress
Name:           nginx-ingress-848796976d-hszvq
Namespace:      nginx-ingress
Priority:      0
Node:          k8s2/10.1.1.5
Start Time:    Fri, 06 Aug 2021 22:10:24 +0000
Labels:        app=nginx-ingress
Annotations:   cni.projectcalico.org/podIP: 192.168.109.75/32
               cni.projectcalico.org/podIPs: 192.168.109.75/32
               prometheus.io/port: 9113
               prometheus.io/scrape: true
Status:        Running
IP:            192.168.109.75
IPs:          IP: 192.168.109.75
Controlled By: ReplicaSet/nginx-ingress-848796976d
Containers:
  nginx-plus-ingress:
    Container ID: docker://d5bb5d75016ecf243a96b2f1f853cafe810760453a71841d776989cf52209ade
    Image:         registry:5000/nginx-plus-kic:1.12.0
    Image ID:     docker-pullable://registry:5000/nginx-plus-kic@sha256:9e1b9db21ecd2237673efde31eeea5a8afe3c7935f451fc3dcf14d972b97ece4
    Ports:        80/TCP, 443/TCP, 8081/TCP, 9000/TCP, 9113/TCP
    Host Ports:   0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP
    Args:
      -nginx-plus
      -nginx-configmaps=$(POD_NAMESPACE)/nginx-config
      -default-server-tls-secret=$(POD_NAMESPACE)/default-server-secret
      -nginx-status-port:9000
      -nginx-status-allow-cidrs=0.0.0.0/0
      -enable-snippets
      -report-ingress-status
      -enable-prometheus-metrics
    State:        Running
```

Check the KIC Nginx Plus Dashboard

1. Using Kubernetes [port-forwarding](#), see if the dashboard is running on port 9000 . Using the VScode terminal pane, run the following `kubectl port-forward` command:

```
kubectl port-forward -n nginx-ingress $KIC_POD_NAME 9000:9000
```

2. Now open Chrome web browser to view the NGINX Plus Dashboard, at <http://localhost:9000/dashboard.html>.

Do you see the NGINX Plus Dashboard? If so, your Ingress Controller pod is up and running!

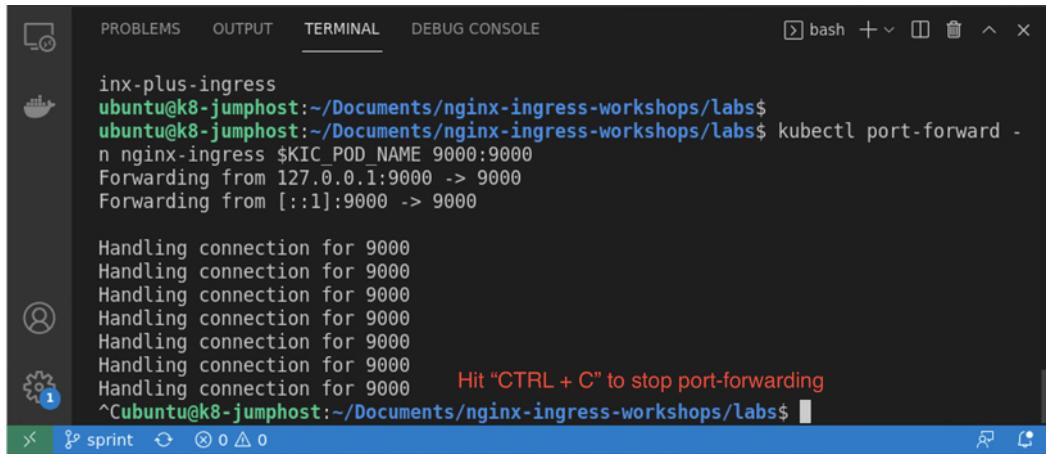
Connections		SSL		Accepted	
Current	Accepted/s	Active	Idle	Dropped	Accepted: 17
6	0	1	5	0	

Requests	
Current	Req/s
1	13

Question - Why is there almost nothing else to see?

- ▶ Click for Hints!

3. Close Chrome Web Browser, and hit `Ctrl-C` in the terminal to stop the Port Forward.



A screenshot of the Visual Studio Code interface showing a terminal window. The terminal tab is selected, displaying the following text:

```
inx-plus-ingress
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl port-forward -n nginx-ingress $KIC_POD_NAME 9000:9000
Forwarding from 127.0.0.1:9000 -> 9000
Forwarding from [::1]:9000 -> 9000

Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Hit "CTRL + C" to stop port-forwarding
^Cubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$
```

The terminal window has a blue status bar at the bottom with icons for sprint, a refresh arrow, and other settings.

Take a look "under the hood" of Ingress Controller

The NGINX Ingress Controller is a pod running Nginx Plus under the hood, let's go check it out.

1. Use the VScode Terminal to enter a shell in the NGINX Ingress Controller pod by running the [kubectl exec](#) command

```
kubectl exec -it $KIC_POD_NAME -n nginx-ingress -- /bin/bash
```

2. Once inside a shell in the NGINX Ingress Controller pod, run the following commands to inspect the root NGINX configuration

```
cd /etc/nginx
more nginx.conf
```

If you have worked with Nginx config files, it should look very similar!

3. Type `q` to quit viewing the `nginx.conf`

```
worker_processes auto;

daemon off;

error_log stderr notice;
pid /var/lib/nginx/nginx.pid;

events {
    worker_connections 1024;
}
```

--More-- (4%)

Type `q` at any time to quit viewing the file

4. Type `exit` to close the connection to the Ingress pod.

```
daemon off;

error_log stderr notice;
pid /var/lib/nginx/nginx.pid;

events {
    worker_connections 1024;
}

nginx@nginx-ingress-848796976d-nq85s:/etc/nginx$ exit
exit
type 'exit' to exit the shell in the pod
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$
```

This completes this Lab.

References:

- <https://docs.nginx.com/nginx-ingress-controller>
- <https://docs.nginx.com/nginx-ingress-controller/installation/installation-with-manifests/>

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab3](#) | [Main Menu](#))

Lab 3: Configuring Nginx Ingress Controller

Introduction

The Nginx Ingress Controller is already running in this Workshop. You will be configuring Ingress Controller for external access outside of your cluster using a `LoadBalancer` Service. The Kubernetes `LoadBalancer` Service is what assigns IP addresses to the `Ingress`, so it can communicate with clients outside the cluster, and also with pods inside the Cluster.

Note: Your Kubernetes cluster is running on a private lab network. We will be using a private IP for external access. In a public cloud environment, the `LoadBalancer` Service from a Cloud Provider would provide you with a routable public IP address.

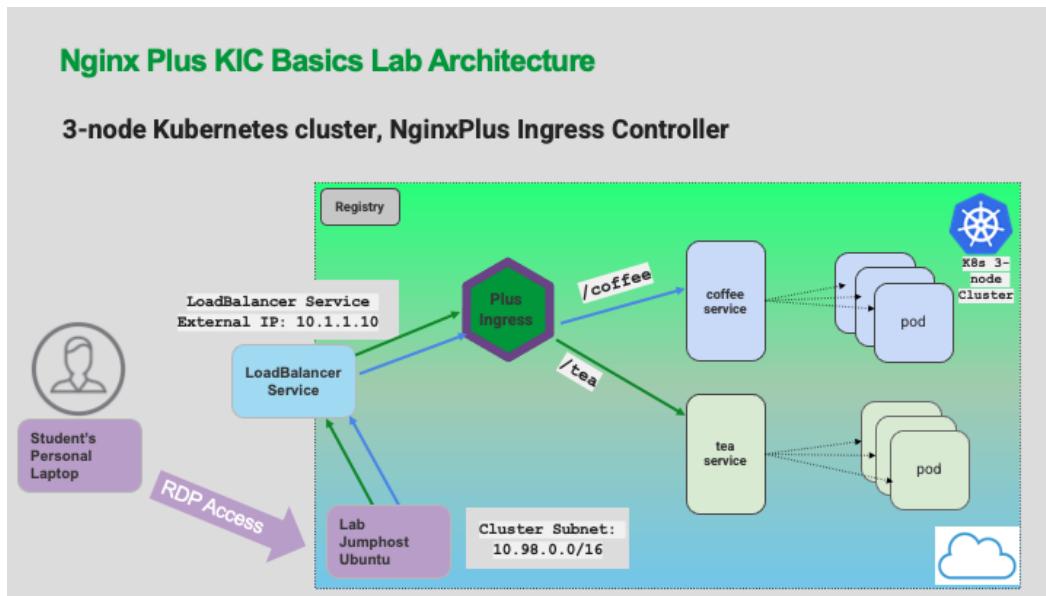
Learning Objectives

- Test and verify the `LoadBalancer` Service
- Test and verify access to the Ingress Controller using the external IP address.

Inspect the LoadBalancer Service

1. Review this Workshop's network architecture diagram below. There are 3 important pieces of information needed for these exercises:

- `LoadBalancer` Service
- An External IP address
- The Kubernetes Cluster IP subnet, used to assign IPs to Pods, including the Ingress Controller



2. Inspect the `lab3/loadbalancer.yaml` manifest. You can see that port `80` and `443` are being opened and we are requesting an external IP address. This will give the Ingress Controller a static private IP address from an IP address management system in the lab.

```

labs > lab3 > ! loadbalancer.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-ingress
5    namespace: nginx-ingress
6  spec:
7    externalTrafficPolicy: Local
8    type: LoadBalancer
9    ports:
10   - port: 80
11     targetPort: 80
12     protocol: TCP
13     name: http
14   - port: 443
15     targetPort: 443
16     protocol: TCP
17     name: https
18   selector:
19     app: nginx-ingress

```

IMPORTANT SECURITY NOTE: In a real world deployment using a Cloud Provider, with a public IP address, this would expose your Ingress Controller to the open Internet with **NO PROTECTION** other than basic TCP port filters. Doing this in production would require Security/Firewall Protections, which are not part of this lab exercise.

3. Confirm there is an `nginx-ingress` service with `TYPE: LoadBalancer`. Run the following command to get networking details of our pod:

```
kubectl get deployments,services -n nginx-ingress
```

You will see the two IP addresses for the Ingress Controller. Both of these IPs must exist for the `LoadBalancer` service to work correctly:

- **EXTERNAL-IP** : This is your external IP address
- **CLUSTER-IP** : This is your Kubernetes internal IP address

```

ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get deployments,services -n nginx-ingress
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress        1/1     1           1           4d
service/service.nginx-ingress       1/1     1           1           25h

NAME          TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)         AGE
service/nginx-ingress   LoadBalancer   10.98.133.36   10.1.1.10      80:31871/TCP,443:32470/TCP   25h

```

In the example above you see:

- Cluster-IP address of `10.98.133.36`
- External-IP address of `10.1.1.10`

- Both IPs are mapped from port `80` to a NodePort (`31871`); and from port `443` to NodePort (`32470`)

NOTE:

- Your `Cluster-IP` address may be different based on your cluster.
- Since this is a lab environment and not a public cloud environment, the `External-IP` address is contained within the lab and has no public access.

Verify access to the Ingress Controller using the External IP

- Use the `LoadBalancer` `External-IP` address that we captured from the previous step to test your `nginx-ingress` service. Use `curl` command to test it.

```
#Test Access to Ingress through LoadBalancer:  
curl -I http://10.1.1.10
```

You should see the following output if the `LoadBalancer` Service is configured correctly for Ingress:

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ curl -I http://10.1.1.10  
HTTP/1.1 404 Not Found  
Server: nginx/1.19.10  
Date: Tue, 10 Aug 2021 22:56:53 GMT  
Content-Type: text/html  
Content-Length: 154  
Connection: keep-alive
```

Question: Why did you get a 404?

- Click for Hints!

This completes this Lab.

References:

- [Nginx KIC LoadBalancer for cloud providers](#)
- [Kubernetes LoadBalancer Service Type](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

This completes the Lab.

Navigate to ([Lab4](#) | [Main Menu](#))

Lab 4: Nginx Plus Dashboard access

Introduction

In this section, you are going to use the NGINX Plus Dashboard to monitor both NGINX Ingress Controller as well as our backend applications. This is a great feature to allow you to watch and triage any potential issues with NGINX Ingress controller as well as any issues with your backend applications, for example, HTTP response times, healthcheck failures, status codes, TCP connections, SSL sessions, etc.

Learning Objectives

- Deploy the Nginx Dashboard Service
- Test access to the Dashboard

Deploy the Nginx Dashboard Service

We will deploy a `Service` and a `VirtualServer` resource to provide access to the NGINX Plus Dashboard for live monitoring. NGINX Ingress [VirtualServer](#) is a [Custom Resource Definition \(CRD\)](#) used by NGINX to configure NGINX Server and Location blocks for NGINX configurations.

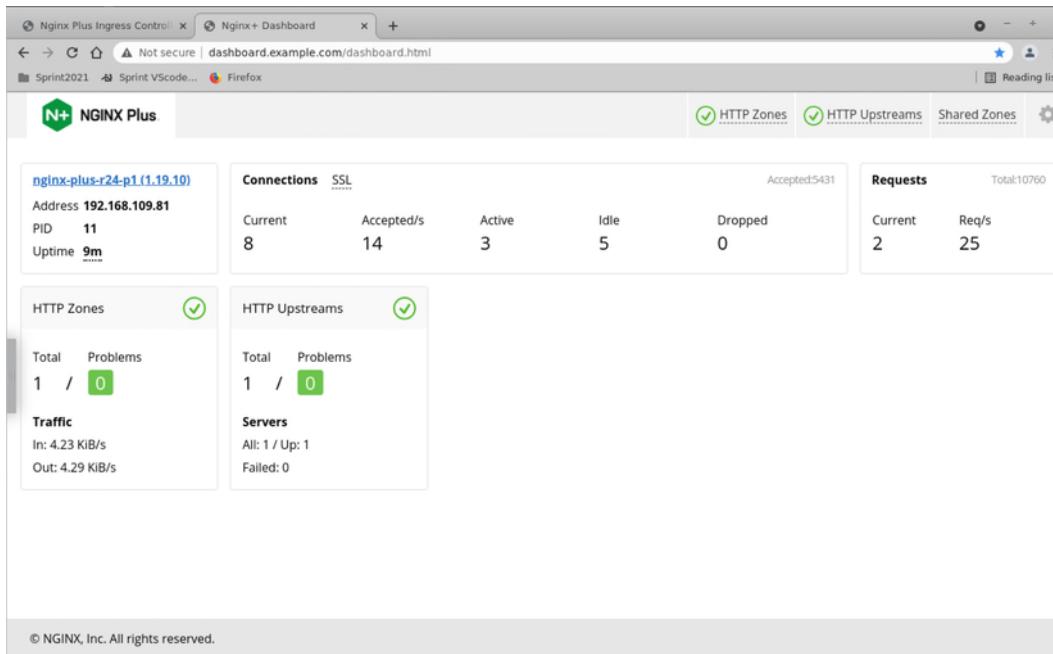
1. In the `lab4` folder, apply the `dashboard-vs.yaml` file to deploy a `Service` and a `VirtualServer` resource to provide access to the NGINX Plus Dashboard for live monitoring

```
kubectl apply -f lab4/dashboard-vs.yaml
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ k apply -f lab4/dashboard-vs.yaml
service/dashboard-svc created
virtualserver.k8s.nginx.org/dashboard-vs created
```

Test access to the Dashboard

1. Open a new Chrome web browser tab, and click the Dashboard Bookmark, or directly open <http://dashboard.example.com/dashboard.html>



You should see the same Nginx Plus Dashboard as the `kubectl port-forward` test we did in a previously. Now your dashboard is exposed outside of your cluster at <http://dashboard.example.com/dashboard.html>.

Recommended: Leave this Dashboard Window open for the rest of the Workshop, you will refer to it often during later exercises.

Congratulations! You have successfully configured your Ingress Controller for external access and the Nginx Plus Dashboard. Next we will deploy some application services and start routing some traffic through Nginx Ingress.

This completes this Lab.

References:

- [NGINX Plus Live Activity Monitoring](#)
- [NGINX Plus Dashboard example](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab5](#) | [Main Menu](#))

Lab 5: Deploy the Nginx Cafe Ingress demo application, using manifests

Introduction

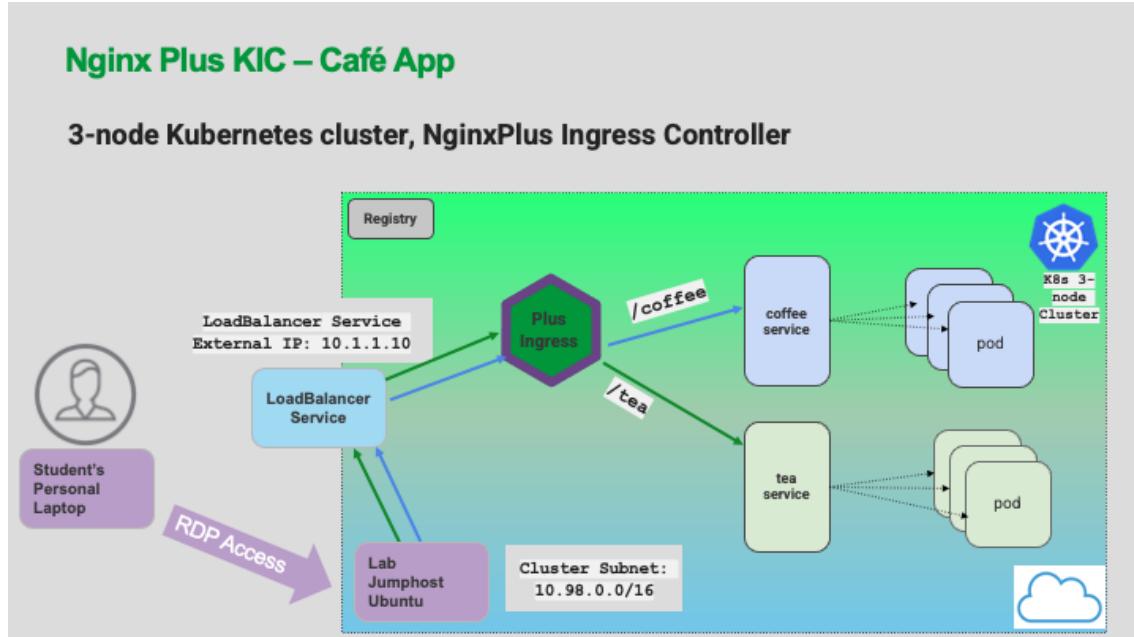
In this section, you will build the "Cafe" Ingress Demo, which represents a Coffee Shop website with Coffee and Tea applications. You will be adding the following components to your Kubernetes Cluster: `Coffee` and `Tea` services , `cafe-secret` , and `cafe virtualserver` .

Learning Objectives

- Deploy the Cafe Demo app
- Compare VirtualServer and Ingress manifests
- Verify the URL path access to `/coffee` and `/tea` work correctly
- Monitor the Nginx Plus Dashboard
- Verify the homepage redirect works correctly

Deploy the Cafe Demo app

The Cafe application that you will deploy looks like the following diagram below. Coffee and Tea pods and services, with Nginx Ingress routing the traffic for `/coffee` and `/tea` routes, using the `cafe.example.com` Hostname, and with TLS enabled. There is also a hidden third service - more on that later!



1. Deploy the Cafe application by applying the three manifests:

```
kubectl apply -f lab5/cafe-secret.yaml  
kubectl apply -f lab5/cafe.yaml  
kubectl apply -f lab5/cafe-virtualserver.yaml
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab5/cafe-secret.yaml
secret/cafe-secret created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab5/cafe.yaml
deployment.apps/coffee created
service/coffee-svc created
deployment.apps/tea created
service/tea-svc created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab5/cafe-virtualserver.yaml
virtualserver.k8s.nginx.org/cafe-vs created
```

2. Check that all pods are running, you should see **three** Coffee and **three** Tea pods:

```
kubectl get pods
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
coffee-5869b8cf4f-5kw25  1/1     Running   0          10m
coffee-5869b8cf4f-crd8w  1/1     Running   0          10m
coffee-5869b8cf4f-m85tk  1/1     Running   0          10m
tea-7dd9798597-jbgw4    1/1     Running   0          10m
tea-7dd9798597-pbs9g    1/1     Running   0          10m
tea-7dd9798597-pjd87    1/1     Running   0          10m
walkthrough-74546bf8f7-4x9p7  1/1     Running   4          5d20h
walkthrough-74546bf8f7-tpmnd  1/1     Running   30         301d
```

Note: Ignore the two walkthrough pods as they are part of the UDF lab environment.

3. Check that the Cafe `VirtualServer` , `cafe-vs` , is running:

```
kubectl get virtualserver cafe-vs
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get virtualserver cafe-vs
NAME      STATE   HOST           IP      PORTS   AGE
cafe-vs   Valid   cafe.example.com   14m
```

Note: The `STATE` should be `Valid` . If it is not, then there is an issue with your yaml manifest file (`cafe-vs.yaml`) . You could also use `kubectl describe vs cafe-vs` to get more information about the `VirtualServer` we just created.

Compare VirtualServer and Ingress manifest

1. In the `lab5` folder, inspect the `cafe.yaml` manifest file. Find the following configuration details:

- **Question:** How many coffee and tea pods are we starting with?
 - ▶ Click for Hints!
- **Question:** What are the two `Service` names ?
 - ▶ Click for Hints!

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: coffee
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: coffee
10   template:
11     metadata:
12       labels:
13         app: coffee
14     spec:
15       containers:
16         - name: coffee
17           image: nginxinc/ingress-demo
18           ports:
19             - containerPort: 80
20 ---
21   apiVersion: v1
22   kind: Service
23   metadata:
24     name: coffee-svc
25   spec:
26     type: ClusterIP
27     clusterIP: None
28     ports:
29       - port: 80
30         targetPort: 80
```

1. Now inspect the `cafe-virtualserver.yaml` file.

- **Question:** What is the hostname ?
 - ▶ Click for Hints!
- **Question:** Are we using SSL ? If so, which certificate ?
 - ▶ Click for Hints!
- **Question:** Are healthchecks enabled ?
 - ▶ Click for Hints!
- **Question:** What URI paths are defined, routing to where ?

► Click for Hints!

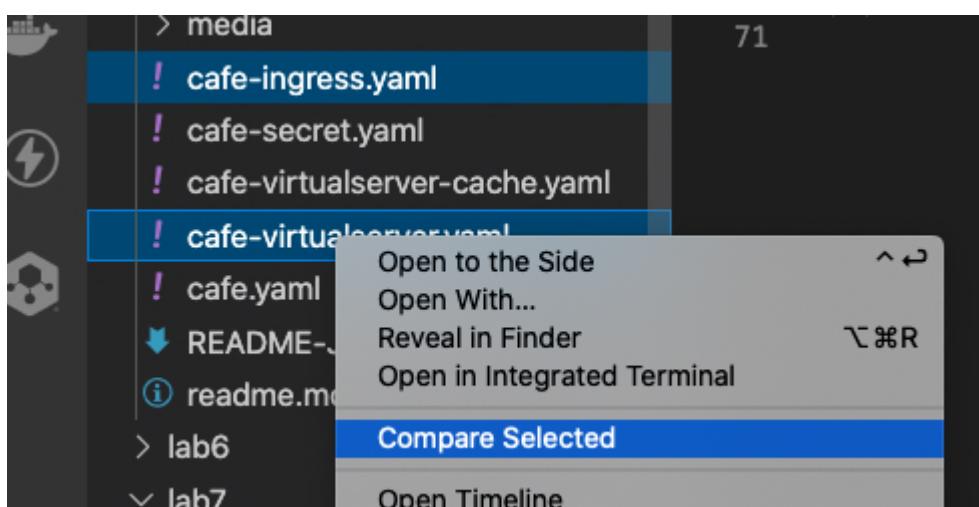
```
3   apiVersion: k8s.nginx.org/v1
4   kind: VirtualServer
5   metadata:
6     name: cafe-vs
7   spec:
8     host: cafe.example.com
9     tls:
10    secret: cafe-secret
11    redirect:
12      enable: true #Redirect from http > https
13      code: 301
14    upstreams:
15      - name: tea
16        service: tea-svc
17        port: 80
18        lb-method: round_robin
19        slow-start: 20s
20    healthCheck:
21      enable: true
22      path: /tea
23      interval: 20s
24      jitter: 3s
25      fails: 5
26      passes: 2
27      connect-timeout: 30s
28      read-timeout: 20s
29      - name: coffee
30        service: coffee-svc
31        port: 80
32        lb-method: round_robin
33    healthCheck:
34      enable: true
35      path: /coffee
36      interval: 10s
37      jitter: 3s
38      fails: 3
39      passes: 2
40      connect-timeout: 30s
41      read-timeout: 20s
```

```
42   routes:
43     - path: /
44       action:
45         rewrite
```

```
45      redirect:
46        url: https://cafe.example.com/coffee
47        code: 302
48      - path: /tea
49        action:
50          pass: tea
51      - path: /coffee
52        action:
53          pass: coffee
54      - path: /milk
55        action:
56          return:
57            code: 200
58            type: text/html
59            body: "Welcome to Nginx Plus KIC Workshop!!"
60
```

1. Compare the `cafe-ingress.yaml` and `cafe-virtualserver.yaml` files. How are they different? Do you see, that the `virtualServer` definition has quite a few more options for controlling how traffic is routed to your Ingress and to your pods:

You can make use of VSCode built-in **compare** tool as seen below: `Select the two files and then right-click Compare Selected:



Do you see how many more options there are in a `VirtualServer` type over a community `Ingress` type?

```

readme.md  ! cafe.yaml  ! cafe-ingress.yaml + cafe-virtualserver.yaml x
lab5 > / cafe-virtualserver.yaml ...
1- apiVersion: networking.k8s.io/v1beta1
2- kind: Ingress

3 metadata:
4- name: cafe-ingress
5 spec:
6- ingressClassName: nginx # use only with k8s version == 1.18.0
7 tls:
8- hosts:
9- - cafe.example.com
10-   secretName: cafe-secret
11-   ports:
12-     - hostPort: 80
13-       hostName: cafe.example.com
14-       https:
15-         paths:
16-           - secret: cafe-secret
17-             redirect:
18-               enabled: true #Redirect from http > https
19-               code: 301
20-               upstream:
21-                 name: tea
22-                 service: tea-svc
23-                 port: 80
24-             healthCheck:
25-               enabled: true
26-               path: /tea
27-               interval: 20s
28-               jitter: 3s
29-               fails: 3
30-               passes: 2
31-               connectTimeout: 30s
32-               readTimeout: 20s
33-             - name: coffee
34-               service: coffee-svc
35-               port: 80

```

- Now inspect the `cafe-secret.yaml` which is the TLS self-signed certificate we are using for this lab.

```

1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: cafe-secret
5  type: kubernetes.io/tls
6  data:
7    tls.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURMa
8    tls.key: LS0tLS1CRUdJTiBSU0EgUFJJVkJURSBLRVktLS0tLQpNS
9

```

Verify the URL path to `/coffee` and `/tea` work correctly

- Access the application using `curl`. We'll use the `-k` option to turn off certificate verification of our self-signed certificate:

```
# To get coffee:
curl -k -I https://cafe.example.com/coffee
```

```
# If you prefer tea:
curl -k -I https://cafe.example.com/tea
```

Monitor the Nginx Plus Dashboard

- Open two new Chrome web browser windows for side by side viewing.

- Dashboard: <http://dashboard.example.com/dashboard.html>:
- And in two tabs, the Cafe Application components, Coffee (<https://cafe.example.com/coffee>) and Tea (<https://cafe.example.com/tea>)

Dashboard

Do you see any changes in the Plus Dashboard? Is `cafe.example.com` in the `HTTPZones` tab? In `HTTP Upstreams` tab do you see **three** coffee and **three** tea pod IP addresses?

The `Server Zones` table contains the Virtual Servers statistics of the Ingress, the `HTTP Upstreams` are the actual Pods running in Kubernetes which are being load balanced.

Zone	Requests			Responses					Traffic			
	Current	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent
dashboard.example.com	1	1022	9	0	1021	0	0	0	1021	6.40 kB	3.60 kB	638 kB
cafe.example.com	0	0	0	0	0	0	0	0	0	0	0	0

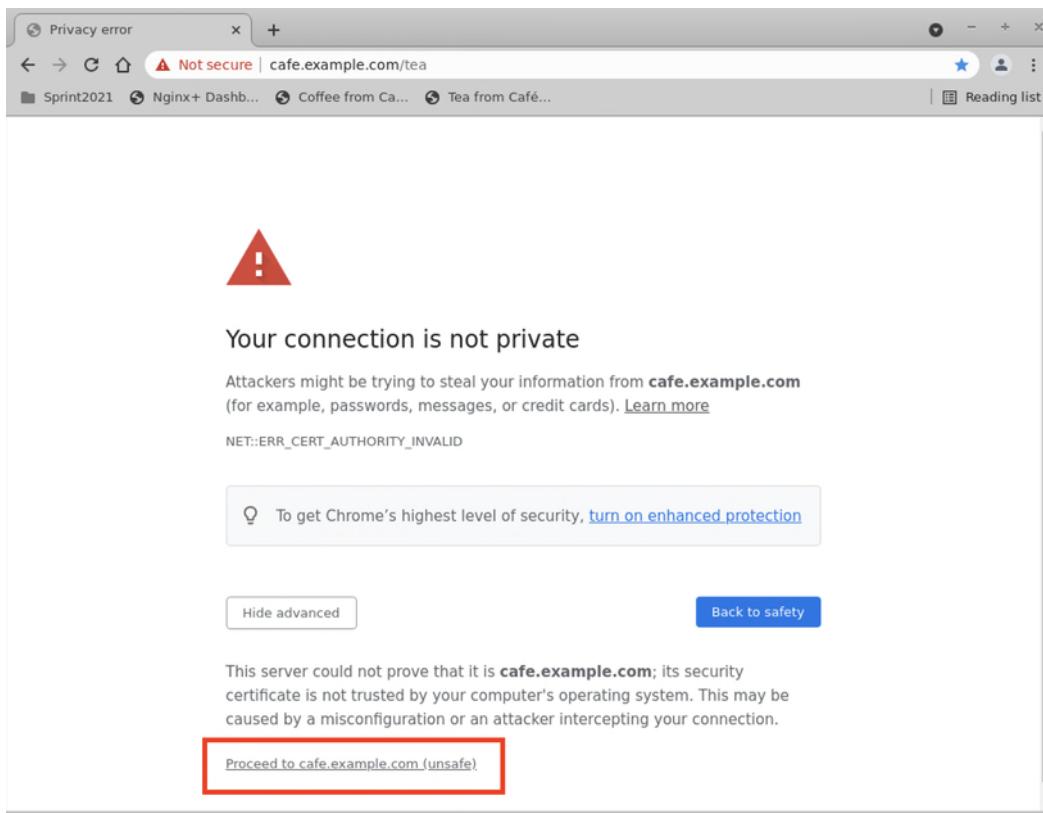
Server	Requests			Responses			Conns		Traffic			Server checks			Health monitors			Response time			
	Name	DT	W	Total	Req/s	-	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers
192.168.26.18:9000	0ms	1	1446	8	>	0	0	1	=	4.13 kB	6.17 kB	734 kB	929 kB	0	0	0	0	0	0	4ms	
vs_default_cafe-vs_coffee	Zone: 0 %																				
192.168.2.92:80	0ms	1	0	0		0	0	0	=	0	0	0	0	0	0	0	49	0	0	passed	-
192.168.27.18:80	0ms	1	0	0	>	0	0	0	=	0	0	0	0	0	0	0	50	0	0	passed	-
192.168.49.193:80	0ms	1	0	0		0	0	0	=	0	0	0	0	0	0	0	51	0	0	passed	-
vs_default_cafe-vs_tea	Zone: 0 %																				
192.168.24.186:80	0ms	1	0	0		0	0	0	=	0	0	0	0	0	0	0	27	0	0	passed	-
192.168.37.136:80	0ms	1	0	0	>	0	0	0	=	0	0	0	0	0	0	0	27	0	0	passed	-
192.168.9.236:80	0ms	1	0	0		0	0	0	=	0	0	0	0	0	0	0	27	0	0	passed	-

Cafe App

1. Using the second Chrome web browser window, open tabs for both:

- Coffee - <https://cafe.example.com/coffee>
- Tea - <https://cafe.example.com/tea>

Did you see an initial Chrome TLS Security warning ? No problem, we are using a self-signed TLS certificate for this Lab and you can safely Proceed.



2. While watching the Dashboard, try refreshing the pages for Coffee and Tea several times:

What do you see in the HTTP Upstreams?

The **Requests** column should increment each time you refresh, and requests are distributed in a "round-robin" Load Balancing method

Upstream	Server	DT	W	Total	Req/s	Responses	Conn	Tr
vs_default_cafe-vs_coffee	192.168.2.92:80	0ms	1	11	0	0 5 6 0 0	0	0
	192.168.27.18:80	0ms	1	11	0	0 5 6 0 0	0	0
	192.168.49.193:80	0ms	1	10	0	0 6 4 0 0	0	0

Upstream	Server	DT	W	Total	Req/s	Responses	Conn	Tr
vs_default_cafe-vs_tea	192.168.24.186:80	0ms	1	8	0	0 3 5 0 0	0	0
	192.168.37.136:80	0ms	1	8	0	0 3 5 0 0	0	0
	192.168.9.236:80	0ms	1	8	0	0 4 4 0 0	0	0

Server Name: coffee-6c689c4f74-hzg49
Server Address: 192.168.27.18:80

NGINX Version: 1.19.6
Request Date: 25/May/2021:16:47:40 +0000
Request URI: /coffee
Document Root: /usr/share/nginx/html/beverage
Ingress Controller IP: 192.168.26.18:51796
Client IP: 98.220.166.131

3. Check the Server Name and Server Address in the gray box of the Coffee and Tea webpage, it should change as you refresh. The Server Name is the Kubernetes assigned pod name, and the Server Address you see is the Pod's internal IP address, assigned by the cluster network.

Do they match up with the Server IPs in the Dashboard?

4. Verify the Endpoint addresses, with `kubectl` commands that shows you the details of Coffee and Tea Service :

```
# Describe Coffee Service
kubectl describe svc coffee-svc

# Describe Tea Service
kubectl describe svc tea-svc
```

The Service Endpoints should match the Server IPs in the dashboard:

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl describe svc coffee-svc
Name:           coffee-svc
Namespace:      default
Labels:         <none>
Annotations:   <none>
Selector:       app=coffee
Type:          ClusterIP
IP Families:  <none>
IP:             None
IPs:            <none>
Port:           http  80/TCP
TargetPort:    80/TCP
Endpoints:     192.168.109.90:80,192.168.109.94:80,192.168.219.26:80
Session Affinity: None
Events:        <none>
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl describe svc tea-svc
Name:           tea-svc
Namespace:      default
Labels:         <none>
Annotations:   <none>
Selector:       app=tea
Type:          ClusterIP
IP Families:  <none>
IP:             None
IPs:            <none>
Port:           http  80/TCP
TargetPort:    80/TCP
Endpoints:     192.168.109.91:80,192.168.219.14:80,192.168.219.24:80
Session Affinity: None
Events:        <none>
```

Verify the homepage redirect works correctly

What happens if you try just plain <http://cafe.example.com>? It should redirect you to TLS secured <https://cafe.example.com/coffee>.

1. Open Chrome Developer Tools: Right Click on the webpage and select `Inspect`.

cafe.example.com/coffee

Server Name:	coffee-bbbb5855b-r2dgw
Server Address:	10.244.0.240:80
NGINX Version:	1.19.6
Request Date:	01/Oct/2021:17:25:31 +0000
Request URI:	/coffee
Document Root:	/usr/share/nginx/html/beverage
Ingress Controller IP:	10.244.0.100:41424
Client IP:	10.244.1.85

Right Click, then Inspect

- Back
- Forward
- Reload
- Save As...
- Print...
- Cast...
- Create QR Code for this Page
- Translate to English
- View Page Source
- Inspect**

Auto Refresh
Request ID: 5b74c72ec6b580760ee6ac42cd4f54de
© NGINX, Inc. 2020

2. Inspect the HTTP Headers: Open the Network Tab > view Headers .

Welcome to Café NGINX | Welcome to Café NGINX | Welcome to Café NGINX

Not Secure | cafe.example.com/coffee

Server Name:	coffee-6c689c4f74-hzg49
Server Address:	192.168.27.18:80
NGINX Version:	1.19.6
Request Date:	25/May/2021:17:04:32 +0000
Request URI:	/coffee
Document Root:	/usr/share/nginx/html/beverage
Ingress Controller IP:	192.168.26.18:45142
Client IP:	98.220.166.131

Auto Refresh
Request ID: 5c1abf1d8f903d8570a15e3556494293
© NGINX, Inc. 2020

Network

Name	Headers	Preview	Response	Initiator	Timing	Cookies
cafe.example.com	Request URL: http://cafe.example.com/ Request Method: GET Status Code: 301 Moved Permanently Remote Address: 34.226.18.190:80 Referrer Policy: strict-origin-when-cross-origin					
coffee.png	Response Headers Connection: keep-alive Content-Length: 169 Content-Type: text/html Date: Tue, 25 May 2021 17:04:32 GMT Location: https://cafe.example.com/ Server: nginx/1.19.5					
	Request Headers Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9					

6 requests 65.3 kB in 1s

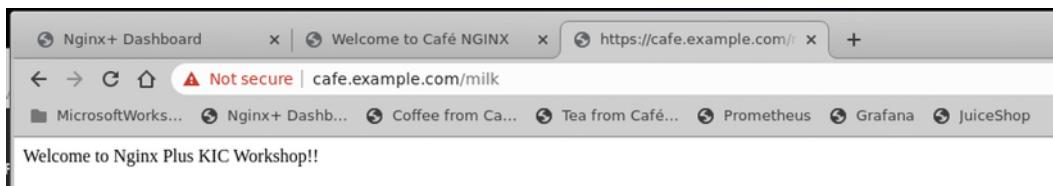
Console What's New X

Highlights from the Chrome 90 update

3. Try <https://cafe.example.com/milk>

Question: What just happened and why?

► Click for Hints!



Now, let's throw some traffic at your Cafe Ingress!

This completes this Lab.

References:

- [Nginx Ingress Controller Complete Example](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab6](#) | [Main Menu](#))

Lab 6: HTTP Load Testing and Cafe Scaling

Introduction

In this lab, you will send some HTTP traffic to your Ingress Controller and Cafe Application, and watch Nginx Ingress load balance the traffic. You will scale the application pods and Ingress Controller up and down, and watch what happens in realtime, as Nginx responds dynamically to these cluster changes.

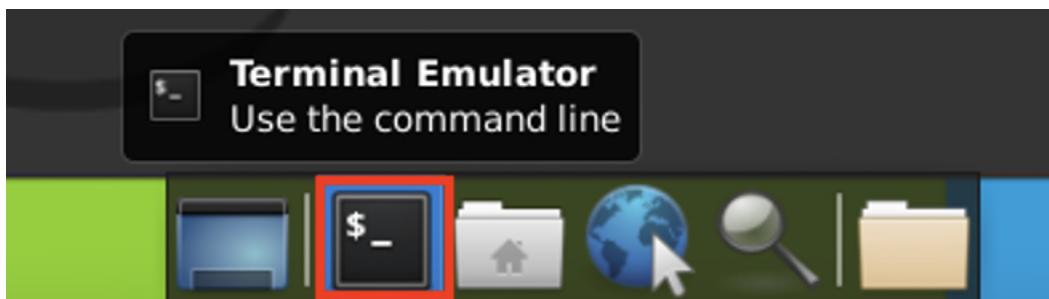
Learning Objectives

- HTTP Traffic Generation
- Scale Applications
- Optional Exercises
- Host-based Routing
- Observe Nginx Dashboard

HTTP Traffic Generation

We will use a tool called `wrk`, running in a docker container, to generate traffic to our Ingress.

1. Open a Terminal from the Ubuntu Desktop



2. In the terminal window, run this command to start load generation using `wrk` inside a docker container:

```
docker run --rm williamyeh/wrk -t4 -c200 -d20m -H 'Host: cafe.example.com' --  
timeout 2s https://10.1.1.10/coffee
```

A screenshot of a terminal window titled "ubuntu@k8-jumphost: ~". The window shows the command being run: "ubuntu@k8-jumphost:~\$ docker run --rm williamyeh/wrk -t4 -c200 -d20m -H 'Host: cafe.example.com' --timeout 2s https://10.1.1.10/coffee". Below this, the output of the command is visible, showing the text "Running 20m test @ https://10.1.1.10/coffee". The terminal has a standard Linux-style interface with a menu bar at the top.

This will run the `wrk` load tool for **20 minutes**, with **200 connections**. You can run this command again if you need additional time.

3. Observe your NGINX Plus Dashboard - <http://dashboard.example.com/dashboard.html>

The screenshot shows the NGINX Plus Dashboard interface. At the top, there's a header with the NGINX Plus logo and navigation links. Below the header, there are three main sections: 'Connections' (highlighted with a red box), 'HTTP Zones', and 'HTTP Upstreams'. The 'Connections' section displays real-time statistics: Current (208), Accepted/s (32), Active (142), Idle (66), Dropped (0), Current (138), and Req/s (1901). A note below the table says 'Counters are incrementing!'. The 'HTTP Zones' and 'HTTP Upstreams' sections show their respective counts and problem status (0 problems). The bottom of the dashboard includes a footer with the copyright notice: '© NGINX, Inc. All rights reserved.'

Questions:

- Do you see the number of HTTP requests increasing?
- Do the coffee pods each take ~equal traffic?
- What does the HTTP Zones table show?
 - ▶ Click for Hints!

Scale Applications

Coffee Break Time !! Let's scale the Coffee Deployment . In anticipation of a surge in coffee drinkers - The boss says we need more Keurig pods.

1. While watching the Dashboard, change the number (replicas) of Coffee pods from **three**, to a total of **eight**:

```
kubectl scale deployment coffee --replicas=8
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl scale deployment coffee --replicas=8
deployment.apps/coffee scaled
```

How long did it take Kubernetes to add the new pods?

Nginx Ingress will run Active healthchecks against the pods, and then begin routing traffic to the **five** new pods. Nginx Ingress Controller automatically discovers the new pods and immediately reconfigures its upstream groups and route traffic to them once marked healthy (Pass the health checks).

Active healthchecks, DNS service Discovery and NGINX Plus reconfiguration API are Premium features of Nginx Plus, these features are used here, where the NGINX Ingress Controller uses the Nginx Plus API to automatically update the list Upstreams with newly discovered pods and begins to load balance them.

Caffiene crises averted...

HTTP Upstreams												HTTP Zones			HTTP Upstreams			Shared Zones		
nginx-ingress-dashboard-ingress-dashboard...												Zone: 8 %			Show all			Failed only		
Server Name	DT	Requests		Responses		Conns		Traffic		Server checks			Health monitors			Response time				
		Total	Req/s	2xx	3xx	4xx	5xx	A	L	Sent/s	Rvd/s	Sent	Rvd	Fails	Unavailable	Checks	Fails	Unhealthy	Last	
192.168.26.18:9000	0ms	1	35100	8	2	0	2	~	3.56 kB	8.45 kB	17.4 MB	28.5 MB	0	0	0	0	0	~	10ms	10ms

vs_default_cafe-vs_coffee												Zone: 0 %			Show all						
Server Name	DT	Requests		Responses		Conns		Traffic		Server checks			Health monitors			Response time					
		Total	Req/s	2xx	3xx	4xx	5xx	A	L	Sent	Rvd	Sent	Rvd	Fails	Unavailable	Checks	Fails	Unhealthy	Last	Headers	Response
192.168.2.92:80	0ms	3236	5	0	3222	14	0	0	0	~ 2.20 kB	8.28 kB	1.40 MB	5.31 MB	0	0	524	0	0	passed	4ms	4ms
192.168.27.18:80	0ms	3236	5	0	3221	15	0	0	0	~ 2.20 kB	8.28 kB	1.40 MB	5.37 MB	0	0	523	0	0	passed	4ms	4ms
192.168.49.193:80	0ms	3236	5	0	3222	14	0	0	0	~ 2.20 kB	8.29 kB	1.40 MB	5.20 MB	0	0	528	0	0	passed	4ms	4ms
192.168.27.107:80	0ms	555	4	0	555	0	0	0	0	~ 1.76 kB	6.63 kB	244 kB	918 kB	0	0	29	0	0	passed	4ms	4ms
192.168.13.160:80	0ms	555	4	0	555	0	0	0	0	~ 1.76 kB	6.63 kB	244 kB	918 kB	0	0	29	0	0	passed	4ms	4ms
192.168.26.218:80	0ms	555	5	0	555	0	0	0	0	~ 2.20 kB	8.29 kB	244 kB	918 kB	0	0	30	0	0	passed	4ms	4ms
192.168.44.148:80	0ms	555	5	0	555	0	0	0	0	~ 2.20 kB	8.29 kB	244 kB	918 kB	0	0	29	0	0	passed	4ms	4ms
192.168.61.87:80	0ms	555	5	0	555	0	0	0	0	~ 2.20 kB	8.28 kB	244 kB	918 kB	0	0	29	0	0	passed	4ms	4ms

2. Inspect the NGINX Plus Dashboard. Look at the far right column of the `HTTP Upstreams` view and you see `HTTP Header` and `Response times` values in milliseconds.

Nginx Plus tracks the actual response time of the pods, when it receives the HTTP header, and when it receives the entire response body. Wouldn't it be nice if you could actually send more requests to the pods responding the fastest? With NGINX Plus, you can do just that.

3. Let's configure NGINX Ingress Controller to use a NGINX Plus Load Balancing algorithm called Least-Time Last-Byte so that the most responsive (faster) pods are prioritized for more HTTP Requests. Apply the following manifest:

```
kubectl apply -f lab6/nginx-config-lastbyte.yaml
```

- **Question:** Why are some pods faster or slower than others?
 - ▶ Click for Hints!

4. After changing the algorithm, now look at the Dashboard HTTP Upstream metrics, you should see the **faster** pods taking more requests than the slower pods.

Note: Faster means lower Response Time in milliseconds.

This is an important NGINX Plus feature. NGINX tracks each pod's response time, and distributes more traffic to faster pods in the cluster.

In today's Digital Experience production workloads, it is important to send customers' requests to the most reliable and performant pod (server). NGINX Plus's LeastTime - LastByte Load Balancing algorithm allows you to handle more total requests, and, it adjusts automatically as pod performance and conditions in the Kubernetes cluster change minute by minute.

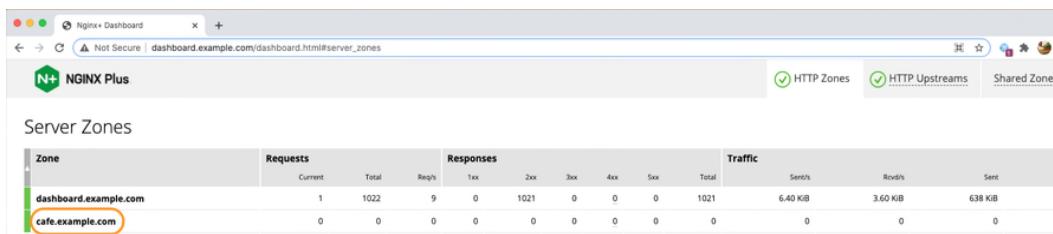
This yields a better customer experience, essential for today's modern digital experience workloads.

NOTE: the Response time differential must be `> 20ms` between the pods, for NGINX to consider some pods faster than others.

5. Boss says Coffee rush is now over, so scale back the number of coffee pods to **three**. Run the following `kubectl scale` command:

```
kubectl scale deployment coffee --replicas=3
```

Did you notice any errors in the NGINX Plus Dashboard while Kubernetes scaled up/down the pods?



When scaling down, NGINX will complete workloads before Kubernetes terminates a running pod , after all the responses in flight have finished processing on that pod , so there should not be any errors. When scaling up, NGINX also healthchecks new pods before sending any requests to them, to make sure they are ready to take requests.

Did you notice any errors when you asked NGINX to change its load balancing algorithm from round-robin to `Least-time` `Last-byte` ? There should not have been any errors. The Dynamic Configuration Reload feature allows existing connections to finish their work, while allowing new connections to use the new configuration.

*This results in **no dropped connections during configuration changes**.*

Optional Lab Exercise1:

Try the same scale up, then scale down commands for the Cafe **Tea** Deployment . Does NGINX also send more traffic to the faster Tea pods? Set the tea pods back to **three** replicas when you are finished.

Note: You will have to target `/tea` with the `wrk` tool if you want to load test Tea:

1. Run the following command to apply load to the Tea Service:

```
docker run --rm williamyeh/wrk -t4 -c200 -d15m -H 'Host: cafe.example.com' --  
timeout 2s https://10.1.1.10/tea
```

Optional Lab Exercise2:

1. You can change the load balancing algorithm back to round-robin if you like, and check out the differences. Apply the following manifest to make that change:

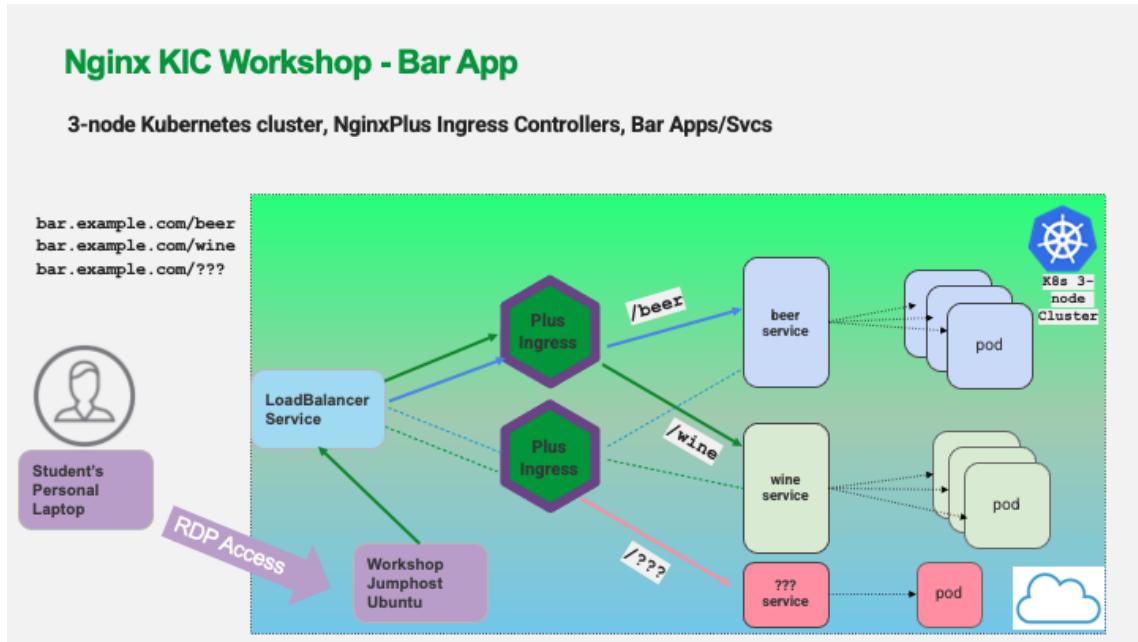
```
kubectl apply -f lab6/nginx-config-roundrobin.yaml
```

Host Based Routing

New business opportunity: The boss got a business loan to open the Cafe as a Bar in the evening for beer and wine tasting! So you need to add Beer and Wine applications, and expose these using the NGINX Ingress Controller.

Hurry up - it's almost Happy Hour!

See the topological view of the new **beer** and **wine** applications



1. In the `lab6` folder, inspect the `bar.yaml` and `bar-virtualserver.yaml` manifest files. You will see that we are adding the required `Deployment` and `Service`, and exposing them with `VirtualServer`, using a new hostname, `bar.example.com`.

```

1  #Example virtual server with routes for Bar Demo
2  #
3  apiVersion: k8s.nginx.org/v1
4  kind: VirtualServer
5  metadata:
6    name: bar-vs
7  spec:
8    host: bar.example.com
9    tls:
10      secret: cafe-secret
11      redirect:
12        enable: true    #Redirect from http > https
13        code: 301
14      upstreams:
15        - name: beer
16          service: beer-svc
17          port: 80
18          slow-start: 20s
19          healthCheck:
20            enable: true
21            path: /beer
22            interval: 20s
23            jitter: 3s
24            fails: 5
25            passes: 2
26            connect-timeout: 30s
27            read-timeout: 20s
28        - name: wine
29          service: wine-svc
30          port: 80
31          #lb-method: round_robin      You can set LB method here if you like
32          healthCheck:
33            enable: true
34            path: /wine
35            interval: 10s
36            jitter: 3s
37            fails: 3
38            passes: 2
39            connect-timeout: 30s

```

2. Create the **beer** and **wine** Deployment and Service , by applying the provided manifests.

```
kubectl apply -f lab6/bar.yaml
```

3. Create a new VirtualServer listening on the hostname, **bar.example.com** , with **/beer** and **/wine** URI paths to the new application deployments, by applying the **bar-virtualserver.yaml** file:

```
kubectl apply -f lab6/bar-virtualserver.yaml
```

4. Open two new Chrome web browser windows and browse to <https://bar.example.com/beer> and <https://bar.example.com/wine>.

Note: You can once again safely proceed to this insecure site.

Do the new `beer` and `wine` applications load as expected?

Request URL	Server Name	Server Address	NGINX Version	Request Date	Document Root	Ingress Controller IP	Client IP
/beer	beer-57fb8c597-ttgsq	192.168.219.11:80	1.19.6	24/Aug/2021:19:48:04 +0000	/usr/share/nginx/html/beverage	192.168.109.73:53352	10.1.1.9
/wine	wine-85554576c7-wc84v	192.168.219.15:80	1.19.6	24/Aug/2021:19:48:06 +0000	/usr/share/nginx/html/beverage	192.168.109.73:38684	10.1.1.9

5. Observe your KIC Plus Dashboard again on <http://dashboard.example.com/dashboard.html>

Questions:

- What does the Dashboard show you when you refresh `/beer` and `/wine` multiple times?
- Did you find your new `HTTP upstreams` for **beer** and **wine**?
- What did you find in the `HTTP Zones` table?

Zone	Requests			Responses					Traffic				
	Current	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcv/s	Sent	Rcvd
bar.example.com	0	16	0	0	16	0	0	0	16	0	0	258 kB	8.58 kB
cafe.example.com	0	0	0	0	0	0	0	0	0	0	0	0	0
dashboard.example.com	1	1232	9	0	1230	0	1	0	1231	11.5 kB	1.34 kB	1.32 MiB	411 kB

The screenshot shows the NGINX Plus Dashboard with five tabs for different Virtual Servers:

- vs.default_bar-vs_beer**: Monitors three servers (IPs 192.168.100.80, 192.168.210.80, 192.168.100.80) with 2 active zones.
- vs.default_bar-vs_cosmo**: Monitors three servers (IPs 192.168.210.80, 192.168.100.80, 192.168.100.80) with 2 active zones.
- vs.default_bar-vs_wine**: Monitors three servers (IPs 192.168.210.80, 192.168.210.80, 192.168.100.80) with 2 active zones.
- vs.default_cafe-vs_coffee**: Monitors three servers (IPs 192.168.100.80, 192.168.100.70, 192.168.210.80) with 2 active zones.
- vs.default_cafe-vs_tea**: Monitors three servers (IPs 192.168.100.70, 192.168.210.80, 192.168.100.80) with 2 active zones.

Each tab displays a table with columns for Server, Requests, Responses, Cache, Traffic, Server checks, Health monitors, and Response time. The tables show various metrics like total requests, errors, and health status for each server.

You can see how easy it is to add new Deployment and Service, and then configure NGINX Plus Ingress Controller to create a new HTTP Host route, URI paths and route traffic to these new services and have access to these applications immediately. The NGINX Plus VirtualServer CRD's allow you to easily add new hosts and services.

This ability of self-service Host and Path based HTTP routing makes it easy to add and deploy new services, and NGINX Plus Ingress routes the traffic correctly (**and BTW - you didn't have to submit even one Ticket to IT and WAIT !!**).

As you can see, this enables application/development teams to rapidly build, deploy, and expose new digital services for the business, without waiting for lengthy deployment/approval processes.

Bonus: Sneaky Pete! There was one additional Happy Hour Drink Special cocktail added by the boss's daughter. Did you find it?

► Click for Answer!

You should also notice that Cafe is still running as before, adding the Bar services and VirtualServer definitions had no impact.

This completes this Lab.

References:

- [Least time Algorithm](#)
- [Random Algorithm](#)
- [Dynamic Reconfiguration with NGINX Plus](#)
- [Wrk Tool](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab7](#) | [Main Menu](#))

Lab 7: Nginx Ingress Controller High Availability and Enhanced Logging

Introduction

In Production Kubernetes clusters, it is important to have High Availability for the Ingress Controllers, just like any traditional load balancer, and the pods themselves. Nginx Ingress Controllers can be added and removed just like application pods.

It is considered a Best Practice to have **at least three Ingress Controllers** for Production workloads running in the cluster. This will provide a very solid High Availability foundation for managing production traffic.

It is also helpful to development and operations teams to see granular details about the traffic to and from the services and pods. NginxPlus has the capability to provide many details to gain insight into these traffic flows, using additional logging variables.

Learning Objectives

- Scaling Nginx Ingress Controller
- Seeing Ingress Under the Hood
- Configuring Enhanced Logging

Scaling Nginx Ingress

Next, let's scale the number of Ingress Controllers pods from one to **three**. This will provide High Availability, and also increase performance and capacity.

1. Run the following `kubectl scale` command:

```
kubectl scale deployment nginx-ingress -n nginx-ingress --replicas=3
```

2. To see the newly created NGINX Plus Ingress Controller pods run the following command in the `nginx-ingress` namespace:

```
kubectl get pods -n nginx-ingress
```

You should see **three** NGINX Plus Ingress Controller pods running

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get pods -n nginx-ingress
NAME                  READY   STATUS    RESTARTS   AGE
nginx-ingress-848796976d-jsvss   1/1     Running   0          102s
nginx-ingress-848796976d-lq2vm   1/1     Running   0          102s
nginx-ingress-848796976d-nq85s   1/1     Running   0          3h47m
```

Questions:

- What happened to your `WRK` loadtest traffic on the Dashboard ?
 - Did it drop by approximately 1/3rd. Why?
- Click for Hints!

3. Now scale the number of NGINX Plus Ingress Controllers up to **four**, in anticipation of a surge of traffic from an overnight Digital Marketing campaign for *free coffee=free caffeine*. Run the following `kubectl scale` command:

```
kubectl scale deployment nginx-ingress -n nginx-ingress --replicas=4
```

```
kubectl get pods -n nginx-ingress
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl scale deployment nginx-ingress -n nginx-ingress --replicas=4
deployment.apps/nginx-ingress scaled
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get pods -n nginx-ingress
NAME           READY   STATUS    RESTARTS   AGE
nginx-ingress-848796976d-7vzkw   1/1     Running   0          17s
nginx-ingress-848796976d-jsvss   1/1     Running   0          9m30s
nginx-ingress-848796976d-lq2vm   1/1     Running   0          9m30s
nginx-ingress-848796976d-nq85s   1/1     Running   0          3h55m
```

What do you observe?

Food for thought - With most Cloud providers, you could use AutoScaling to do this automatically!

4. The Marketing push is over, so scale the Ingress Controllers back down to **one**. Run the following `kubectl scale` command:

```
kubectl scale deployment -n nginx-ingress nginx-ingress --replicas=1
```

```
kubectl get pods -n nginx-ingress
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl scale deployment -n nginx-ingress nginx-ingress --replicas=1
deployment.apps/nginx-ingress scaled
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get pods -n nginx-ingress
NAME           READY   STATUS    RESTARTS   AGE
nginx-ingress-848796976d-7vzkw   0/1     Terminating   0          88s
nginx-ingress-848796976d-jsvss   0/1     Terminating   0          10m
nginx-ingress-848796976d-lq2vm   0/1     Terminating   0          10m
nginx-ingress-848796976d-nq85s   1/1     Running   0          3h56m
```

Ingress Under the Hood

1. Let's take peek under the hood – check out the NGINX Plus Ingress Controller Configurations. Run the following commands to view the NGINX Configuration:

```
# Store the KIC Pod name in a variable
export KIC_POD_NAME=$(kubectl get pods -n nginx-ingress -o
jsonpath='{.items[0].metadata.name}')

# Check the full Nginx config
kubectl exec -it $KIC_POD_NAME -n nginx-ingress -- nginx -T
```

Inspect the output: `nginx -T` prints out the entire Nginx configuration. Scroll up and down - do you see:

- server block
- the listen ports
- café.example.com Hostname
- TLS configurations
- upstream blocks with Pod IPs
- the least_time or round-robin load balancing method

This is all standard Nginx Plus under the hood, it should look very familiar.

Enhanced Logging with NGINX Plus Ingress

Let's add some additional fields to the Nginx Access Log, you need more data about the performance of the **coffee** and **tea** pods for Developers. They are asking you to help fix an "**application too slow**" escalation ticket from the Marketing team. The "free coffee" campaign was popular but customer feedback was the website was too slow.

For reference, this is the default Nginx Access Log format:

```
log_format main $remote_addr - $remote_user [$time_local] $request $status
$body_bytes_sent$http_refererm$http_user_agent$http_x_forwarded_for;
```

However, there are only **two** log variables with any useful data related to the actual pods:

- HTTP status code (\$status)
- Bytes Sent (\$body_bytes_sent)

1. Use the `kubectl log` command, to view the default Nginx Plus Ingress Controller Access log format:

```
kubectl logs $KIC_POD_NAME -n nginx-ingress --tail 10 --follow
```

```
ubuntu@k8s-jumphost:~/Documents/nginx-ingress-workshops/labss kubectl logs $KIC_POD_NAME -n nginx-ingress --tail 10 --follow
10.1.1.9 - - [24/Aug/2021:20:36:26 +0000] "GET /coffee/default.js HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:26 +0000] "GET /coffee/coffee.png HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee HTTP/1.1" 200 672 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee/default.css HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee/default.js HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee/coffee.png HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee HTTP/1.1" 200 671 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
10.1.1.9 - - [24/Aug/2021:20:36:27 +0000] "GET /coffee/default.css HTTP/1.1" 304 0 "https://cafe.example.com/coffee" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-"
```

How do you even know *which pod* sent the response? To properly troubleshoot and identify the poor performance of a pod, you need much more information.

2. Type `Ctrl-C` to stop the log `tail` when finished.

3. Lets implement an **Enhanced** Access Log format, to collect extra Nginx Plus Request and Response and pod statistics, we can do this by adding new log variables specific to NGINX Plus Ingress such as the Kubernetes pods' resource and traffic details.

NGINX Plus has many variables that can be used for logging. In the code snippet below (`lab7/nginx-config-enhanced-logging.yaml`), you can see the new **Enhanced** Access Log

format for the NGINX Plus Ingress Controller, as a `ConfigMap` :

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-config
  namespace: nginx-ingress
data:
  lb-method: "least_time last_byte"
  log-format: '$remote_addr - $remote_user [$time_local] "$request" $status
$body_bytes_sent \"$http_referer\" \"$http_user_agent\" \"$http_x_forwarded_for\" 
rn=\"$resource_name\" \"$resource_type\" \"$resource_namespace\" svc=\"$service\" 
$request_id" rt=\"$request_time\" ua=\"$upstream_addr\" 
uct=\"$upstream_connect_time\" uht=\"$upstream_header_time\" 
urt=\"$upstream_response_time\" uqt=\"$upstream_queue_time\" 
cs=\"$upstream_cache_status\"'
```

Taking advantage of the additional logging variables from NGINX Plus will provide detailed insight into which upstream pods are working the best, and which are having issues. This should help the app dev team to identify the slow pods and further help in their troubleshooting steps.

4. Apply the **Enhanced** Access Log format (`lab7/nginx-config-enhanced-logging.yaml`) manifest using the `kubectl apply` command:

```
kubectl apply -f lab7/nginx-config-enhanced-logging.yaml -n nginx-ingress
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab7/nginx-config-enhanced-logging.yaml -n nginx-ingress
configmap/nginx-config created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$
```

5. Let's generate new traffic by refreshing the `cafe.example.com/coffee` webpage in the Chrome web browser several times, to send some requests, to see the new **Enhanced** Access Logging format.

Now we are ready to inspect the the new logs: the Pod's HTTP Header and Response times have been added to the log format, in addition to the pod's actual Kubernetes name. The response time values in the log should be similar to what you observed in the Plus Dashboard.

6. Take a look at the **Enhanced** NGINX Access log format using the `kubectl log` command:

```
kubectl logs $KIC_POD_NAME -n nginx-ingress --tail 10 --follow
```

```

ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl logs skIC_POD_NAME -n nginx-ingress -t tail 10 --follow
10.1.1.9 - [24/Aug/2021:20:22:39 +0000] "GET /api/6/slabs/ HTTP/1.1" 200 2878 "http://dashboard.example.com/dashboard.html" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-" rn="dashboard-vs" "virtualserver" "nginx-ingress" svc="dashboard-svc" "e130c3ce169e5901fc12b7837fcb78b1" rt="0.001" ua="192.168.109.73:9000" uct="0.000" uht="0.000" urt="0.000" uq="0.000" cs="-"
10.1.1.9 - [24/Aug/2021:20:22:39 +0000] "GET /api/6/resolvers/ HTTP/1.1" 200 2 "http://dashboard.example.com/dashboard.html" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-" rn="dashboard-vs" "virtualserver" "nginx-ingress" svc="dashboard-svc" "de9632ec4ebc448352c2ea60676165c4" rt="0.001" ua="192.168.109.73:9000" uct="0.000" uht="0.000" urt="0.000" uq="0.000" cs="-"
10.1.1.9 - [24/Aug/2021:20:22:39 +0000] "GET /api/6/http/server_zones/ HTTP/1.1" 200 488 "http://dashboard.example.com/dashboard.html" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-" rn="dashboard-vs" "virtualserver" "nginx-ingress" svc="dashboard-svc" "iae331e0c60756aeabd5aedd2c0aae08" rt="0.000" ua="192.168.109.73:9000" uct="0.000" uht="0.004" urt="0.004" uqt="0.000" cs="-"
10.1.1.9 - [24/Aug/2021:20:23:39 +0000] "GET /api/6/http/location_zones/ HTTP/1.1" 200 2 "http://dashboard.example.com/dashboard.html" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-" rn="dashboard-vs" "virtualserver" "nginx-ingress" svc="dashboard-svc" "9b6be84df4ed09e9a7c9d5f176e9194" rt="0.001" ua="192.168.109.73:9000" uct="0.000" uht="0.000" urt="0.000" uqt="0.000" cs="-"
10.1.1.9 - [24/Aug/2021:20:23:39 +0000] "GET /api/6/http/upstreams/ HTTP/1.1" 200 6303 "http://dashboard.example.com/dashboard.html" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36" "-" rn="dashboard-vs" "virtualserver" "nginx-ingress" svc="dashboard-svc" "ed99e31e084a91df67bd5c34f870e2e8" rt="0.001" ua="192.168.109.73:9000" uct="0.000" uht="0.000" urt="0.000" uqt="0.000" cs="-"

```

7. Type `Ctrl-C` to stop the log `tail` when finished.

This completes this Lab.

References:

- [Summary of ConfigMap Keys - logging](#)
- [NGINX Variables](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab8](#) | [Main Menu](#))

Monitoring NGINX Plus Ingress with Prometheus and Grafana

Introduction

This lab exercise is going to walk you through how to install and use several tools to monitor your NGINXPlus Ingress Controller in your Kubernetes cluster.

Learning Objectives

By the end of the lab, you will be able to:

- Learn and Use Helm Charts
- Deploy Prometheus using Helm
- Deploy Grafana using Helm
- Access these apps thru NGINX Ingress Controller

Helm	Prometheus	Grafana
		

Here is a brief description of what these different tools and application provide, and how you will use them.

`Helm` will make it simple to install everything into your cluster. This keeps the setup easier to deploy and easier to manage. (Note, you can also install using the `manifests` method or `operator` method as well). It will come down to personal preference or specific requirements when installing software into Kubernetes. You will use the Helm Chart provided by NGINX for this lab exercise.

`Prometheus` is a software package that can watch and collect statistics from many different k8s pods and services. It then provides those statistics in a simple html/text format, often referred to as the "scraper page", meaning that it scrapes the statistics and presents them as a simple text-based web page.

`Grafana` is a data visualization tool, which contains a time series database and graphical web presentation tools. Grafana imports the Prometheus scraper page statistics into its database, and allows you to create `Dashboards` of the statistics that are important to you. There are a large number of pre-built dashboards provided by both Grafana and the k8s community, so there are many available to use. And of course, you can customize them as needed or build your own.

Helm Installation



Note: Helm should already be installed on the Ubuntu Jumpbox for you.

1. Verify Helm is installed and you are running Version 3.x or higher:

```
# check your helm version (Needs to be v3 or higher)
helm version --short
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm version --short
v3.7.0+geeac838
```

If Helm is not installed, run this command to install it:

```
# Install helm command
curl -sSL https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

2. Create a new Kubernetes namespace called `monitoring`. You will use this namespace for Prometheus and Grafana components:

```
kubectl create namespace monitoring
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl create namespace monitoring
namespace/monitoring created
```

Prometheus Installation



1. The first step will be to deploy `Prometheus` into our cluster. Below are the steps to install it using Helm as follows:

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts

helm repo add kube-state-metrics https://kubernetes.github.io/kube-state-
metrics

helm repo update
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm repo add prometheus-community https://prometheus-community
.github.io/helm-charts
d kube-state-metrics https://kubernetes.github.io/kube-state-metrics

helm repo update"prometheus-community" has been added to your repositories
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm repo add kube-state-metrics https://kubernetes.github.io/
kube-state-metrics
" kube-state-metrics" has been added to your repositories
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kube-state-metrics" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helm-ing!*
```

2. Once the repos have been added to Helm, the next step is to deploy a `release`. For this lab, you will create a release called `nginx-prometheus`.

```
helm install nginx-prometheus prometheus-community/prometheus -n monitoring
```

```

ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm install nginx-prometheus prometheus-community/prometheus -n monitoring
NAME: nginx-prometheus
LAST DEPLOYED: Thu Oct 7 20:42:47 2021
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
nginx-prometheus-server.monitoring.svc.cluster.local

Get the Prometheus server URL by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=server" -o jsonpath=".items[0].metadata.name")
kubectl --namespace monitoring port-forward $POD_NAME 9090

The Prometheus alertmanager can be accessed via port 80 on the following DNS name from within your cluster:
nginx-prometheus-alertmanager.monitoring.svc.cluster.local

Get the Alertmanager URL by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=alertmanager" -o jsonpath=".items[0].metadata.name")
kubectl --namespace monitoring port-forward $POD_NAME 9093
#####
##### WARNING: Pod Security Policy has been moved to a global property. #####
##### use .Values.podSecurityPolicy.enabled with pod-based #####
##### annotations #####
##### (e.g. .Values.nodeExporter.podSecurityPolicy.annotations) #####
#####

The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
nginx-prometheus-pushgateway.monitoring.svc.cluster.local

Get the PushGateway URL by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=pushgateway" -o jsonpath=".items[0].metadata.name")
kubectl --namespace monitoring port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/

```

Grafana Installation



1. Next step will be to setup and deploy Grafana into your cluster:

```
helm repo add grafana https://grafana.github.io/helm-charts
```

```

ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm repo add grafana https://grafana.github.io/helm-charts
"grafana" has been added to your repositories

```

2. The Grafana repo is added via Helm. Next you will install Grafana using the below command.
For this lab, you will create a second release called `nginx-grafana`.

```
helm install nginx-grafana grafana/grafana -n monitoring
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm install nginx-grafana grafana/grafana -n monitoring
NAME: nginx-grafana
LAST DEPLOYED: Thu Oct 7 20:47:30 2021
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:
  kubectl get secret --namespace monitoring nginx-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
2. The Grafana server can be accessed via port 80 on the following DNS name from within your cluster:
  nginx-grafana.monitoring.svc.cluster.local
  Get the Grafana URL to visit by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace monitoring -l app.kubernetes.io/name=grafana,app.kubernetes.io/instance=nginx-grafana -o jsonpath='{.items[0].metadata.name}')
  kubectl --namespace monitoring port-forward $POD_NAME 3000
3. Login with the password from step 1 and the username: admin
#####
##### WARNING: Persistence is disabled!!! You will lose your data when #####
#####           the Grafana pod is terminated. #####
#####
```

If you want to check the status of your helm installations, you can run this command which will show all helm deployments across the cluster"

```
helm ls -A
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ helm ls -A
NAME          NAMESPACE   REVISION  UPDATED             STATUS      CHART        APP VERSION
nginx-grafana  monitoring   1         2021-10-07 20:47:30 741107581 +0000 UTC deployed  grafana-6.16.12  8.1.6
nginx-prometheus monitoring   1         2021-10-07 20:42:47.941163425 +0000 UTC deployed  prometheus-14.9.0 2.26.0
```

Testing the NginxPlus Prometheus "scraper" Port and Page

Verify that NginxPlus KIC is enabled for exporting Prometheus statistics. This requires 3 settings:

- Prometheus Annotations are enabled
- Port 9113 is open
- The Plus command-line argument `--enable-prometheus-metrics` is enabled, to allow the collection of the KIC's statistics on that port.

!Bonus! - These settings have already been enabled for you in this lab, but they are not enabled by default.

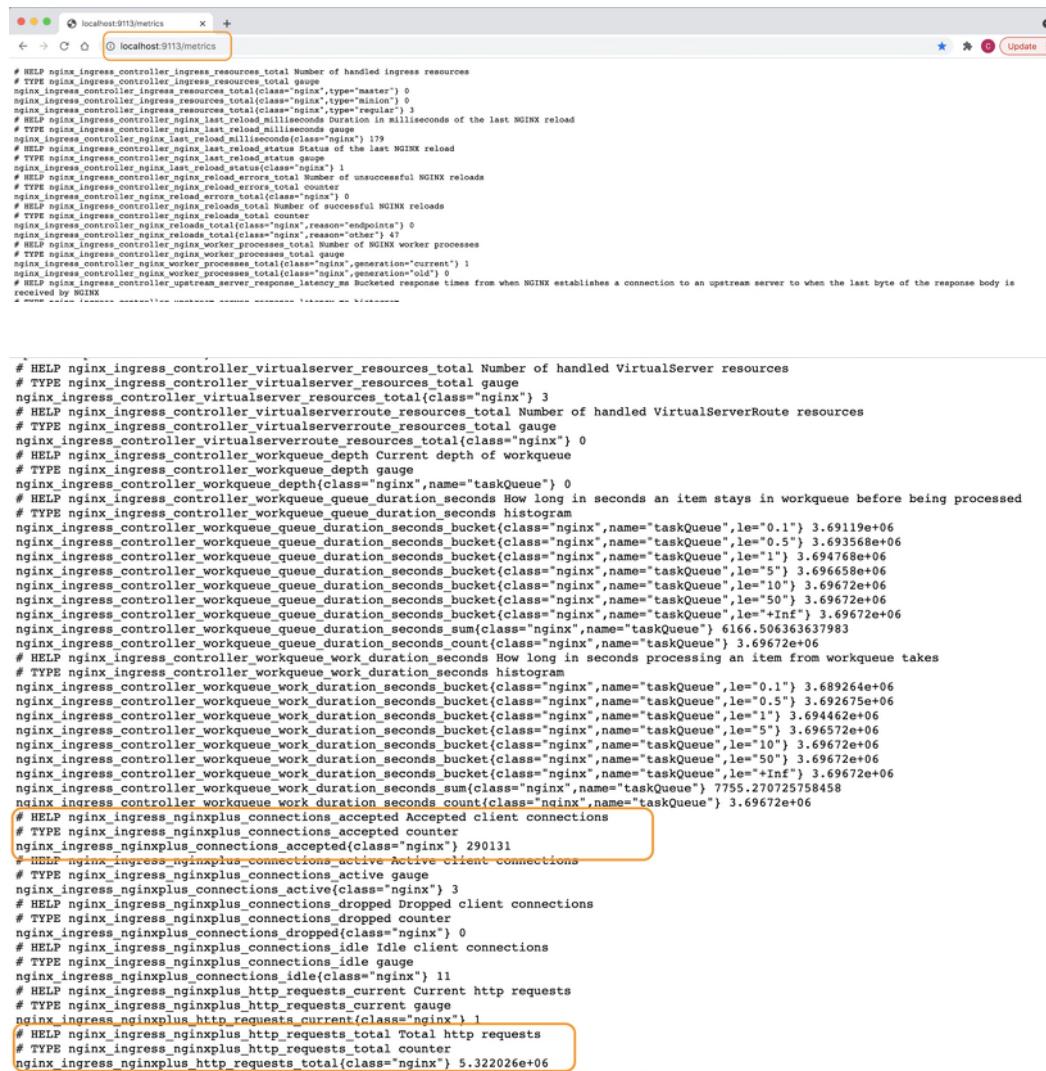
To see these settings, inspect the `lab2/nginx-plus-ingress.yaml` file, lines 15-17, 33-34, and 68.

Annotations	Port	Plus Args
<pre>! nginx-plus-ingress.yaml labs > lab8 > ! nginx-plus-ingress.yaml 1 apiVersion: apps/v1 2 kind: Deployment 3 metadata: 4 name: nginx-ingress 5 namespace: nginx-ingress 6 spec: 7 replicas: 1 8 selector: 9 matchLabels: 10 app: nginx-ingress 11 templates: 12 metadata: 13 labels: 14 app: nginx-ingress 15 annotations: 16 prometheus.io/scrape: "true" 17 prometheus.io/port: "9113" 18 spec:</pre>	<pre>! nginx-plus-ingress.yaml labs > lab8 > ! nginx-plus-ingress.yaml 23 name: nginx-plus-ingress 24 ports: 25 - name: http 26 containerPort: 80 27 - name: https 28 containerPort: 443 29 - name: readiness-port 30 containerPort: 8001 31 - name: dashboard 32 containerPort: 9080 33 - name: prometheus 34 containerPort: 9113</pre>	<pre>! nginx-plus-ingress.yaml labs > lab8 > ! nginx-plus-ingress.yaml 57 args: 58 - --nginx-plus 59 - --nginx-configmaps=\$(POD_NAMESPACE)/nginx-config 60 - --default-server-tls-secret=\$(POD_NAMESPACE)/default-server-secret 61 - --nginx-status-port=9000 62 - --nginx-status-allow-cidrs=0.0.0.0/0 63 - --enable-snippets 64 #- --enable-app-protect 65 #- --v=3 # Enables extensive logging. Useful for troubleshooting. 66 - --report-ingress-status 67 #- --external-service=nginx-ingress 68 - --enable-prometheus-metrics 69 #- --global-configuration=\$(POD_NAMESPACE)/nginx-configuration</pre>

- Now verify this is enabled and working, using k8s port-forward:

```
kubectl port-forward -n nginx-ingress $KIC_POD_NAME 9113:9113
```

Open Chrome, and navigate to <http://localhost:9113/metrics>. You should see an HTML scraper page like this one:



```
# HELP nginx_ingress_controller_ingress_resources_total Number of handled ingress resources
# TYPE nginx_ingress_controller_ingress_resources_total gauge
nginx_ingress_controller_ingress_resources_total{class="nginx",type="master"} 0
nginx_ingress_controller_ingress_resources_total{class="nginx",type="minion"} 0
nginx_ingress_controller_ingress_resources_total{class="nginx",type="regular"} 3
# HELP nginx_ingress_controller nginx_last_reload_milliseconds Duration in milliseconds of the last NGINX reload
# TYPE nginx_ingress_controller nginx_last_reload_milliseconds gauge
nginx_ingress_controller.nginx.last.reload.milliseconds(class="nginx") 179
# HELP nginx_ingress_controller nginx_last_reload_status Status of the last NGINX reload
# TYPE nginx_ingress_controller nginx_last_reload_status gauge
nginx_ingress_controller.nginx.last.reload.status(class="nginx") 1
# HELP nginx_ingress_controller nginx_reload_errors_total Number of unsuccessful NGINX reloads
# TYPE nginx_ingress_controller nginx_reload_errors_total counter
nginx_ingress_controller.nginx.reload.errors.total(class="nginx") 0
# HELP nginx_ingress_controller nginx_reloads_total Number of successful NGINX reloads
# TYPE nginx_ingress_controller nginx_reloads_total counter
nginx_ingress_controller.nginx.reloads.total(class="nginx",reason="endpoints") 0
nginx_ingress_controller.nginx.reloads.total(class="nginx",reason="other") 47
# HELP nginx_ingress_controller.nginx_worker_processes_total Total number of NGINX worker processes
# TYPE nginx_ingress_controller.nginx_worker_processes_total gauge
nginx_ingress_controller.nginx.worker_processes.total(class="nginx",generation="current") 1
nginx_ingress_controller.nginx.worker_processes.total(class="nginx",generation="old") 0
# HELP nginx_ingress_controller_upstream_server_response_latency_ms Bucketed response times from when NGINX establishes a connection to an upstream server to when the last byte of the response body is received by NGINX
# HELP nginx_ingress_controller_virtualserver_resources_total Number of handled VirtualServer resources
# TYPE nginx_ingress_controller_virtualserver_resources_total gauge
nginx_ingress_controller.virtualserver_resources_total{class="nginx"} 3
# HELP nginx_ingress_controller_virtualserverroute_resources_total Number of handled VirtualServerRoute resources
# TYPE nginx_ingress_controller_virtualserverroute_resources_total gauge
nginx_ingress_controller.virtualserverroute_resources_total{class="nginx"} 0
# HELP nginx_ingress_controller_workqueue_depth Current depth of workqueue
# TYPE nginx_ingress_controller_workqueue_depth gauge
nginx_ingress_controller.workqueue_depth(class="nginx",name="taskQueue") 0
# HELP nginx_ingress_controller_workqueue_queue_duration_seconds How long in seconds an item stays in workqueue before being processed
# TYPE nginx_ingress_controller_workqueue_queue_duration_seconds histogram
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="0.1") 3.69119e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="0.5") 3.693568e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="1") 3.694768e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="5") 3.696658e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="10") 3.69672e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="50") 3.69672e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.bucket(class="nginx",name="taskQueue",le="+Inf") 3.69672e+06
nginx_ingress_controller.workqueue.queue.duration.seconds.sum(class="nginx",name="taskQueue") 6166.506363637983
nginx_ingress_controller.workqueue.queue.duration.seconds.count(class="nginx",name="taskQueue") 3.69672e+06
# HELP nginx_ingress_controller_workqueue_work_duration_seconds How long in seconds processing an item from workqueue takes
# TYPE nginx_ingress_controller_workqueue_work_duration_seconds histogram
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="0.1") 3.689264e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="0.5") 3.692675e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="1") 3.694462e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="5") 3.696572e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="10") 3.69672e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="50") 3.69672e+06
nginx_ingress_controller.work.duration.seconds.bucket(class="nginx",name="taskQueue",le="+Inf") 3.69672e+06
nginx_ingress_controller.work.duration.seconds.sum(class="nginx",name="taskQueue") 7755.270725758458
nginx_ingress_controller.workqueue.work.duration.seconds.count(class="nginx",name="taskQueue") 3.69672e+06
# HELP nginx_ingress_nginxplus_connections_accepted Accepted client connections
# TYPE nginx_ingress_nginxplus_connections_accepted counter
nginx_ingress_nginxplus_connections_accepted(class="nginx") 290131
# HELP nginx_ingress_nginxplus_connections_active Active client connections
# TYPE nginx_ingress_nginxplus_connections_active gauge
nginx_ingress_nginxplus_connections_active(class="nginx") 3
# HELP nginx_ingress_nginxplus_connections_dropped Dropped client connections
# TYPE nginx_ingress_nginxplus_connections_dropped counter
nginx_ingress_nginxplus_connections_dropped(class="nginx") 0
# HELP nginx_ingress_nginxplus_connections_idle Idle client connections
# TYPE nginx_ingress_nginxplus_connections_idle gauge
nginx_ingress_nginxplus_connections_idle(class="nginx") 11
# HELP nginx_ingress_nginxplus_http_requests_current Current http requests
# TYPE nginx_ingress_nginxplus_http_requests_current gauge
nginx_ingress_nginxplus.http.requests.current(class="nginx") 1
# HELP nginx_ingress_nginxplus_http_requests_total Total http requests
# TYPE nginx_ingress_nginxplus_http_requests_total counter
nginx_ingress_nginxplus.http.requests.total(class="nginx") 5.322026e+06
```

If you see an HTML page from NGINX KIC similar to the one above, you are good to go. If you refresh this page several times, you will see that the stats are immediately updated. Notice that there is a `# TYPE` and `# HELP` in the scraper page entries, which describes the data type and definition of each metric.

It is important to point out, that the scraper page contains all of the NGINX Plus statistics that you see on the Dashboard, and a few extras.

This rich set of metrics will provide a great data source for the monitoring and graphing of NGINX Plus KIC.

There are many tools that can collect, display, alert, and archive these metrics. You will use Prometheus and Grafana next to do just that.

Press `Control + C` to stop the port-forward when you are finished with the scraper page.

Prometheus Testing



1. To test Prometheus you will run this command to create a shell variable for the Prometheus Server:

```
export PROMETHEUS_SERVER=$(kubectl get pods --namespace monitoring -l
"app=prometheus,component=server" -o jsonpath=".items[0].metadata.name")
```

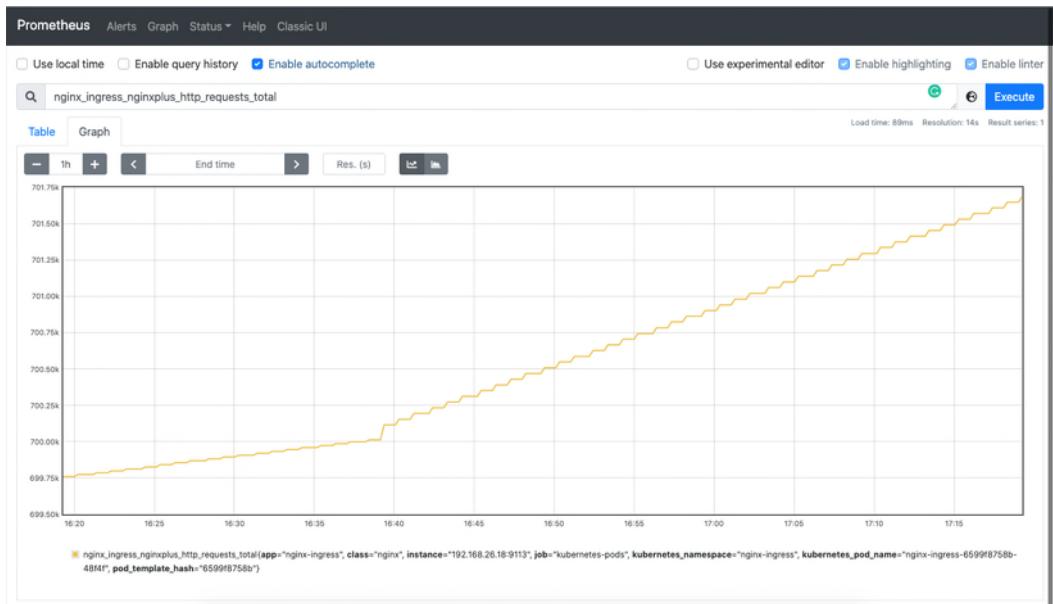
2. Use k8s port-forward to test access to the Prometheus Server:

```
kubectl port-forward $PROMETHEUS_SERVER 9090:9090 -n monitoring
```

Using Chrome, navigate to <http://localhost:9090>. You should see a webpage like this. Search for `nginx_nginxplus_` in the query box to see a list of all the statistics that Prometheus is collecting for you:

A screenshot of the Prometheus web interface. At the top, there's a navigation bar with links for Prometheus, Alerts, Graph, Status, Help, and Classic UI. Below the navigation is a search bar with a placeholder "nginx_nginxplus_". To the right of the search bar are several checkboxes: "Use local time", "Enable query history", "Enable autocomplete" (which is checked), "Use experimental editor", "Enable highlighting", and "Enable linter". Below the search bar is a list of metric names under the heading "METRIC NAMES". The list includes various metrics starting with "nginx_nginxplus_": "connections_accepted", "connections_active", "connections_dropped", "connections_idle", "http_requests_current", "http_requests_total", "server_zone_discarded", "server_zone_processing", "server_zone_received", "server_zone_requests", "server_zone_responses", "server_zone_sent", "ssl_handshakes", "ssl_handshakes_failed", "ssl_session_reuses", "nginxplus_up", "upstream_keepalives", and "upstream_server_active". One metric, "http_requests_total", is highlighted with a blue selection bar. On the far right of the interface, there are buttons for "Remove Panel" and "Execute".

Select `nginx_nginxplus_http_requests_total` from the list, click on Graph, and then click the "Execute" Button. This will provide a graph similar to this one:



Take a few minutes to explore the many different statistics, time windows, etc.

Optional Prometheus Exercise

1. Create a Prometheus graph showing the nginxplus HTTP upstream response time from the pods. This should match the upstream response time values you see in the NGINX Plus Dashboard.

Can you find where the pod CPU stats are recorded ? As you can see, there are hundreds of stats you can use to monitor NGINX KIC, k8s components, pods, and other resources.

Press `Control + C` to stop port-forward when you are finished.

Grafana Testing



1. Retrieve the Grafana admin login password, which was dynamically created by Helm during the installation:

```
kubectl get secret --namespace monitoring nginx-grafana -o jsonpath=".data.admin-password" | base64 --decode ; echo
```

2. To test Grafana, set a shell variable to the Grafana Server to the name of the Grafana pod:

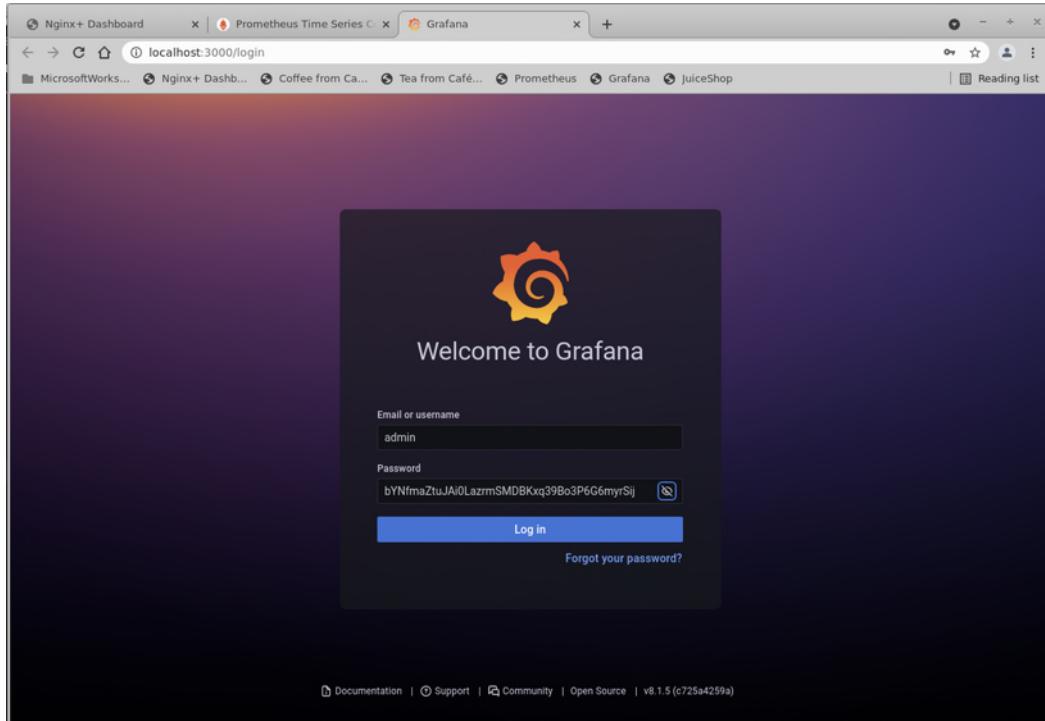
```
export GRAFANA_SERVER=$(kubectl get pods --namespace monitoring -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=nginx-grafana" -o jsonpath=".items[0].metadata.name")
```

3. Use k8s port-forward:

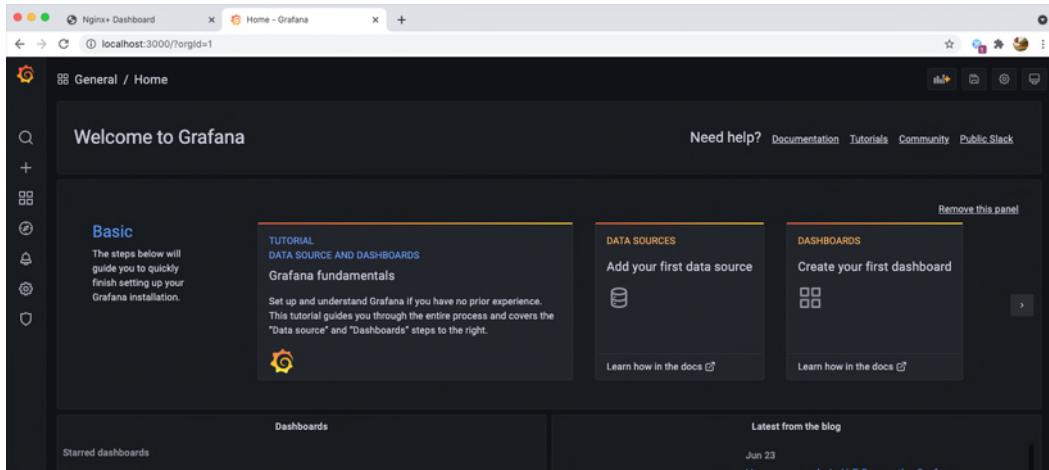
```
kubectl -n monitoring port-forward $GRAFANA_SERVER 3000:3000
```

4. Using Chrome, navigate to <http://localhost:3000> on your browser.

You can login into Grafana using `admin` and the `password` retrieved above.



After logging in, you should see the main Grafana Welcome page:



You will be creating and using some custom Grafana Dashboards in the next sections.

Press `Control + C` to stop port-forward when you are finished.

NGINX Ingress for Prometheus and Grafana

For you and your team to access Prometheus and Grafana from outside the cluster, you will add these apps to your existing NGINX Ingress Controller. In your lab environment, you will use VirtualServer and VirtualServerRoute manifests, to take advantage of NGINX's ability to do cross-namespace routing. (Prometheus and Grafana are running in the "monitoring" namespace, remember?).

1. Inspect the `prometheus-vs.yaml`, `grafana-vs.yaml` and `grafana-vsr.yaml` files in the `lab8` folder.

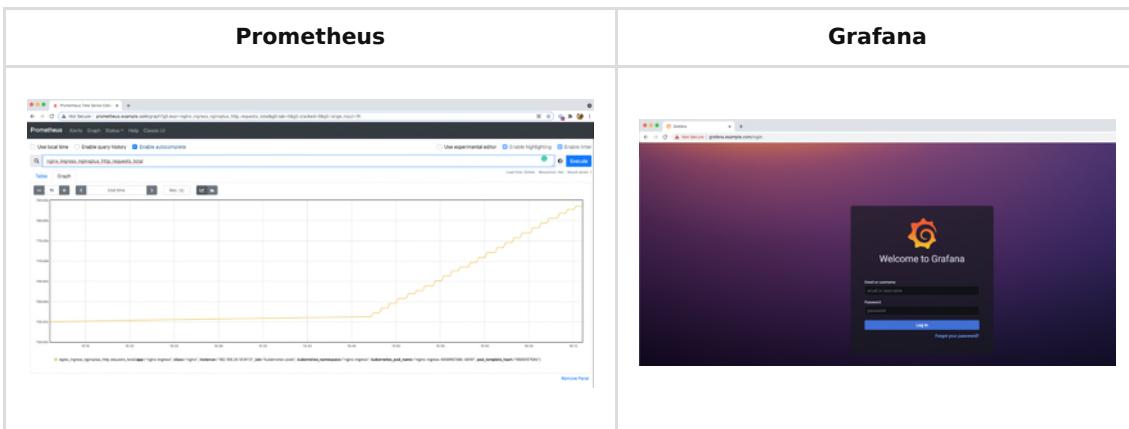
Notice that you are routing requests to your Prometheus and Grafana applications that live in a different namespace (`monitoring` in this lab), which are specified in your `VirtualServer/VirtualServerRoute` configuration.

2. Apply the VS manifests:

```
kubectl apply -f lab8/prometheus-vs.yaml
kubectl apply -f lab8/grafana-secret.yaml
kubectl apply -f lab8/grafana-vs.yaml
kubectl apply -f lab8/grafana-vsr.yaml
```

```
^Cubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab8/prometheus-vs.yaml
apply -f lab8/grafana-secret.yaml
kubectl apply -f lab8/grafana-vs.yaml
kubectl apply -f lab8/grafana-vsr.yamlvirtualserver.k8s.nginx.org/prometheus-vs created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab8/grafana-secret.yaml
secret/grafana-secret created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab8/grafana-vs.yaml
virtualserver.k8s.nginx.org/grafana-vs created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab8/grafana-vsr.yaml
virtualserverroute.k8s.nginx.org/grafana-dashboard created
```

3. To test access through NGINX Ingress, open Chrome and navigate to Prometheus (<http://prometheus.example.com>) and Grafana (<https://grafana.example.com>) bookmarks.

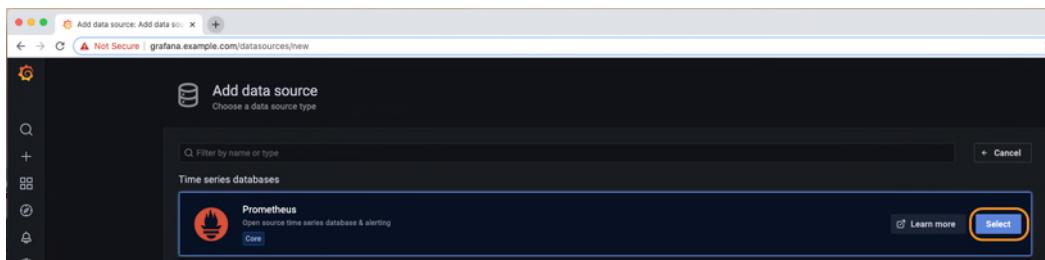


Try another Prometheus query that interests you.

You can login to Grafana using the same admin/password credentials that you used earlier.

Configure Grafana Data Sources

- Once logged in, from the left panel you need to click on Configuration -> Data sources and add Prometheus as a data source.



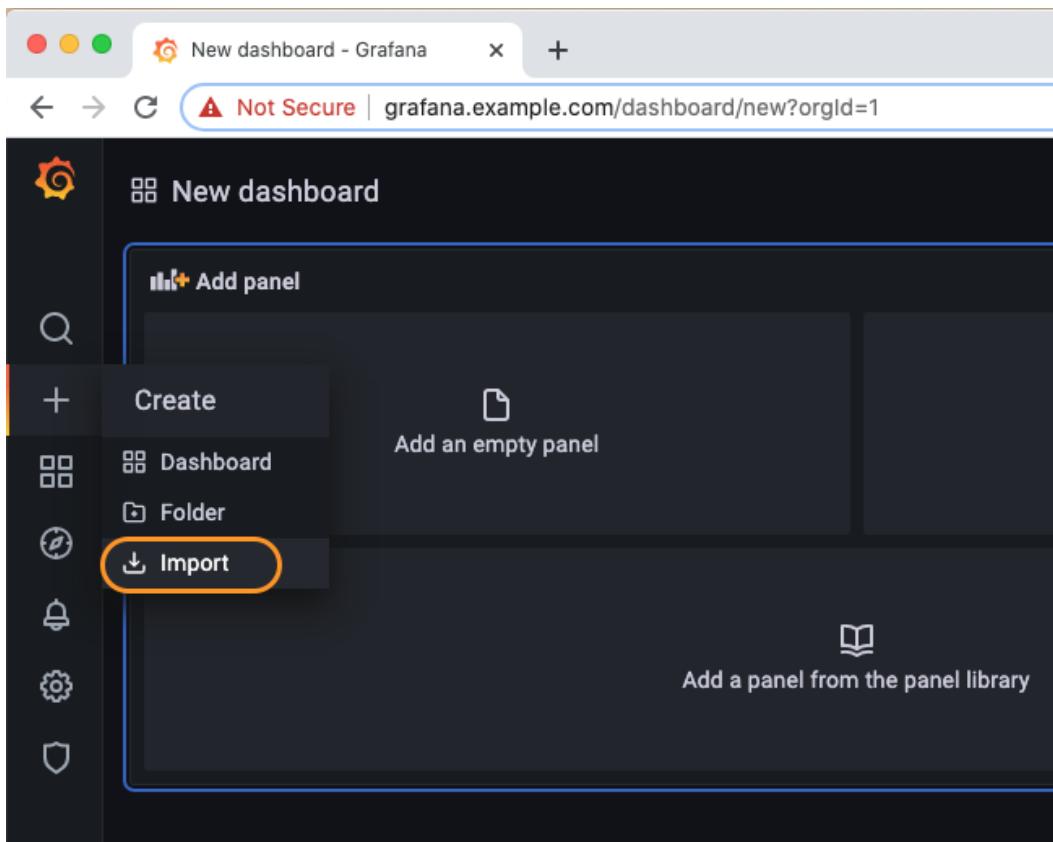
- Once Prometheus is added as a data source, in the Prometheus settings tab, update HTTP URL to nginx-prometheus-server:80 .

The screenshot shows the 'Data Sources / Prometheus' configuration page in Grafana. The left sidebar has 'Configuration' selected, with 'Data sources' highlighted. The main area shows a 'Settings' tab selected. A data source named 'Prometheus' is listed, with its URL set to 'nginx-prometheus-server:80'. The 'HTTP' section includes fields for Access (set to 'Server (default)'), Whitelisted Cookies (with a placeholder 'New tag (enter key to add)'), and Timeout. The 'Auth' section includes options for Basic auth, TLS Client Auth, Skip TLS Verify, and Forward OAuth Identity, all of which are currently disabled (indicated by greyed-out checkboxes). A 'Save and Test' button is located at the bottom right of the form.

Scroll to the bottom, and click "Save and Test", it should show a Green Checkmark status and then click the "Back" button.

Import Grafana Custom Dashboards

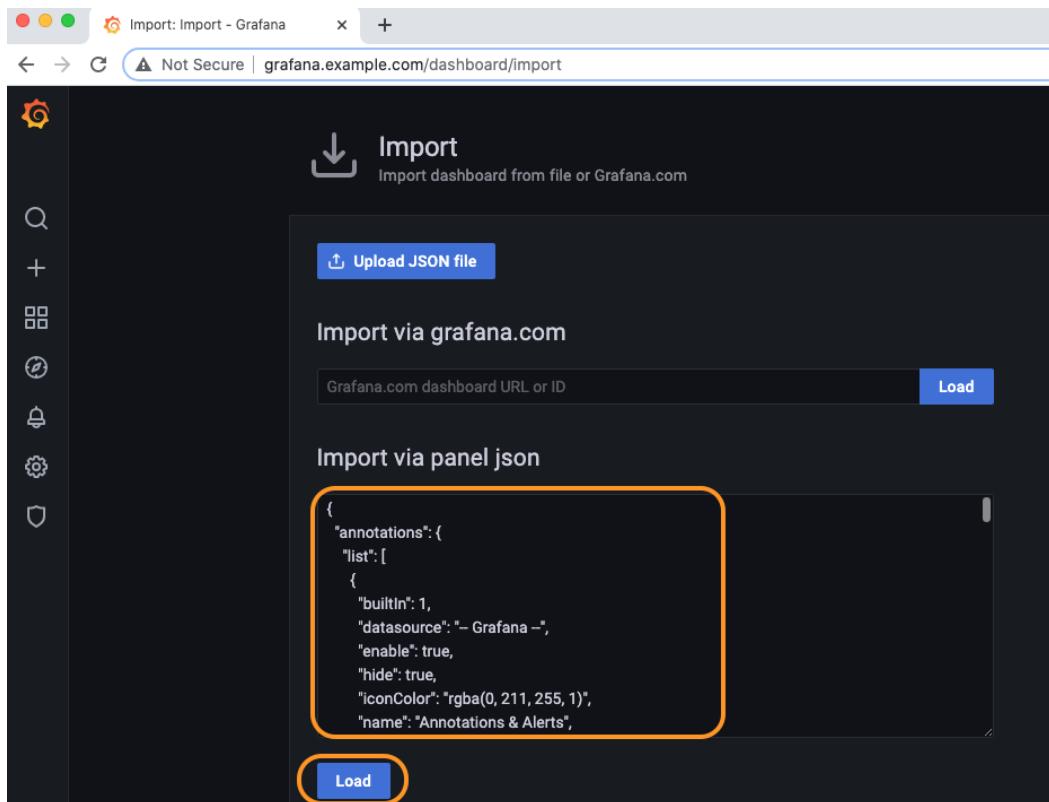
1. Now you should be ready to import the NGINX Dashboards for KIC from NGINX, Inc. From the left panel click on `Import` to add the dashboard:



2. Next you will import the two Grafana dashboard JSON definition files present in the `lab8` folder.

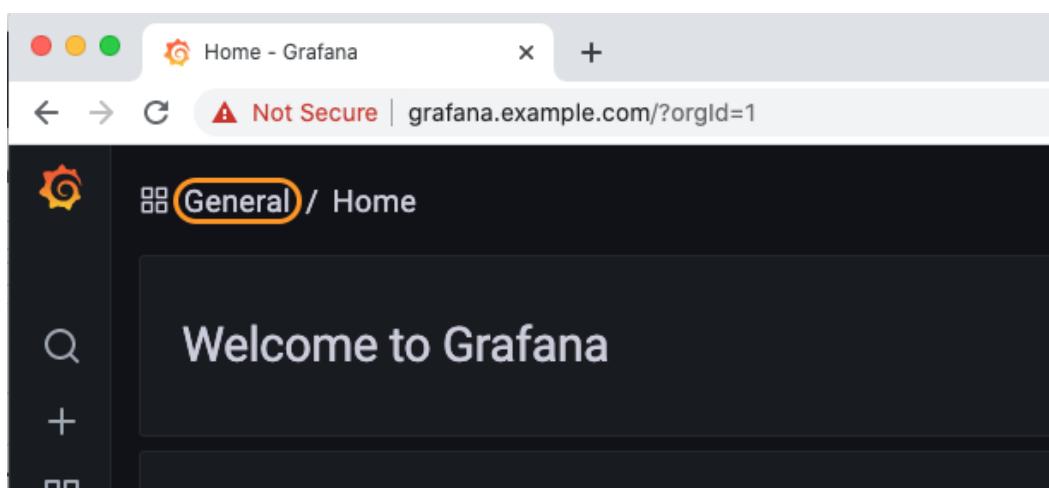
- `NGINX-Basic.json` gives you basic metrics which come from NGINX Opensource.
- `NGINXPlusICDashboard.json` is provided by NGINX, Inc, giving you advanced Layer 4 thru 7 TCP/HTTP/HTTPS metrics which are only available from NGINX Plus.

Copy the entire json file and place it within the `Import via panel json` textbox and click on `Load` button.

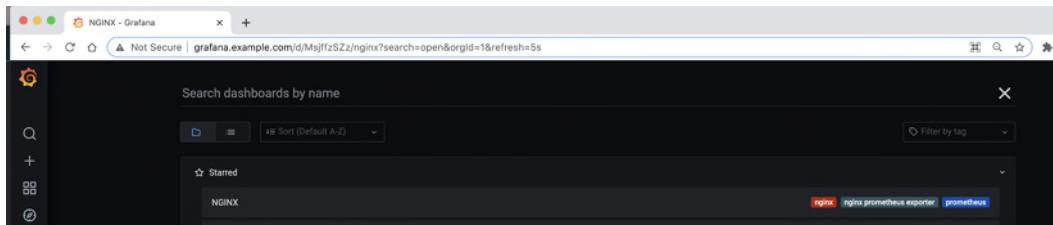


3. Once you have imported both Dashboards, it's time to check them out:

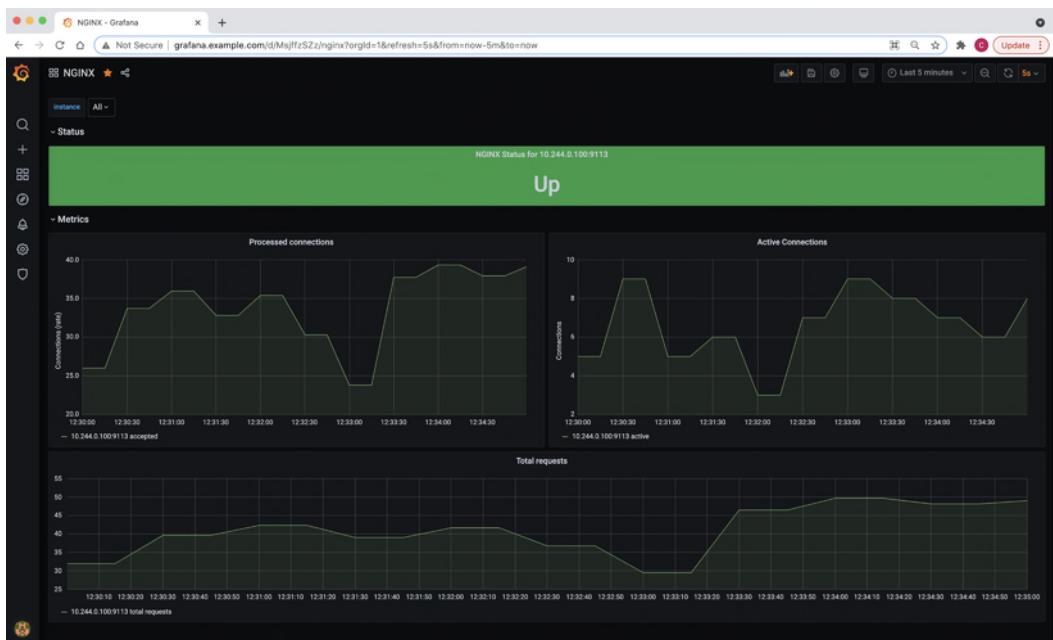
From the Grafana homepage, navigate to the `General` section as shown:



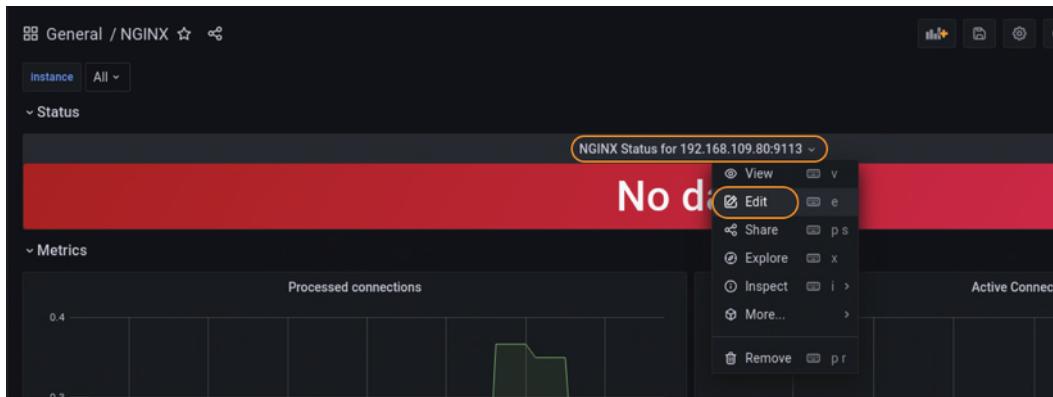
In the `General` section, click on the `NGINX` dashboard.

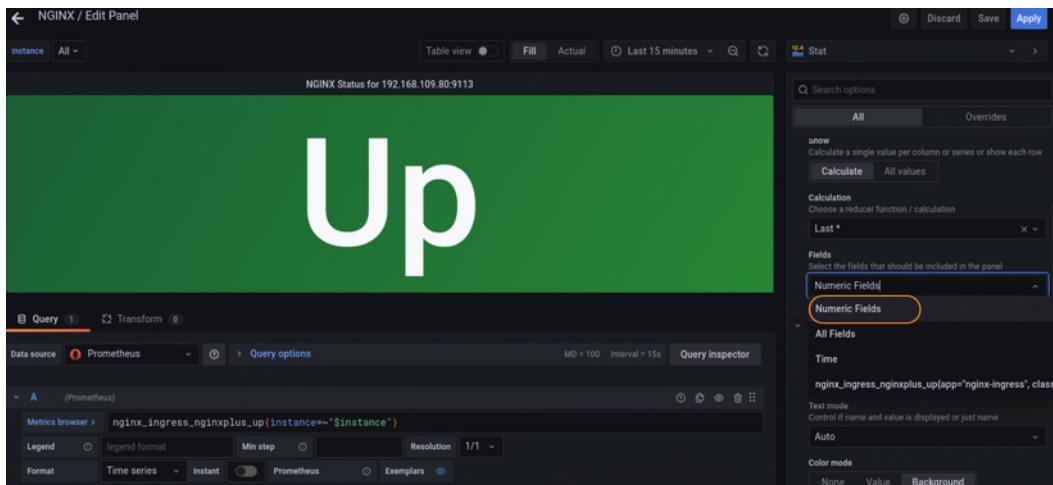


This should open up the NGINX Basic Grafana Dashboard. You can expand the sub-sections or adjust the time range and refresh interval in the upper right corner as needed. You can see this shows the up/down Status of the Ingress, and few Connections and Requests stats:

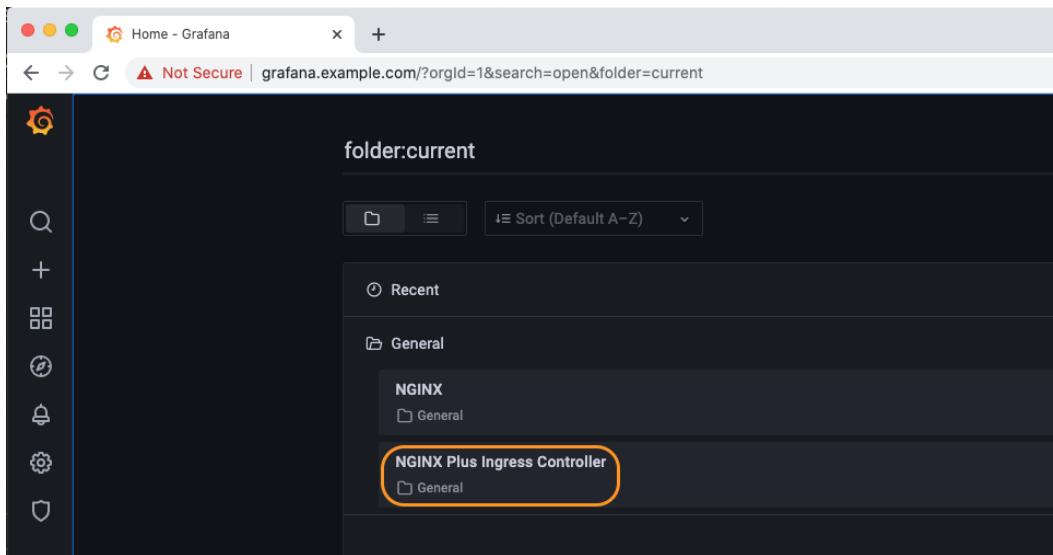


NOTE: If you see a red bar with the message "**No Data**" in the top most pane as seen in below screenshot then click on edit, and change Value options Fields to "Numeric Fields" and click Apply.

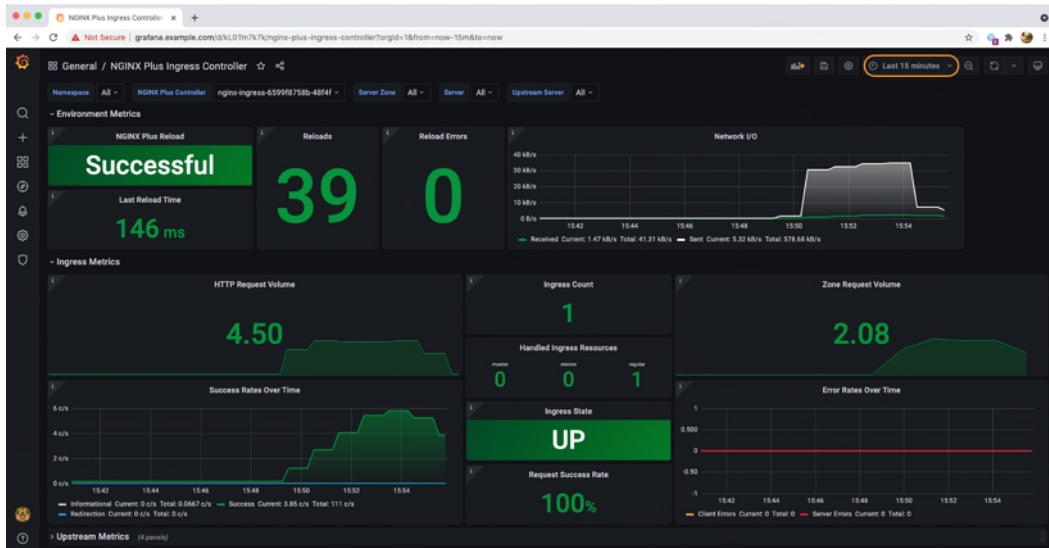




4. Next, from the `General` section, select the `NGINX Plus Ingress Controller Dashboard`.

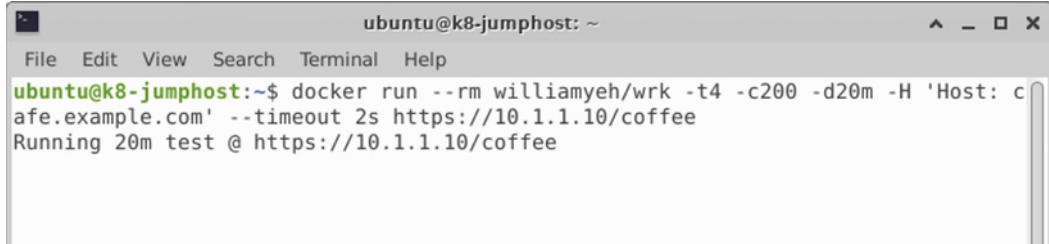


This should open up the NginxPlus Grafana Dashboard. You can expand the sub-sections or adjust the time range and refresh time as needed.



If the graphs are blank or do not show much data, try restarting the loadtest tool from the previous lab, you should see some statistics being collected and graphed after a few minutes:

```
docker run --rm williamyeh/wrk -t4 -c200 -d20m -H 'Host: cafe.example.com' --timeout 2s https://10.1.1.10/coffee
```



This completes this Lab.

References:

- [VirtualServer and VirtualServerRoute](#)
- [Grafana NGINX Plus IC Dashboard](#)

Authors

- Jason Williams - Product Management Engineer @ F5, Inc.
- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab9](#) | [Main Menu](#))

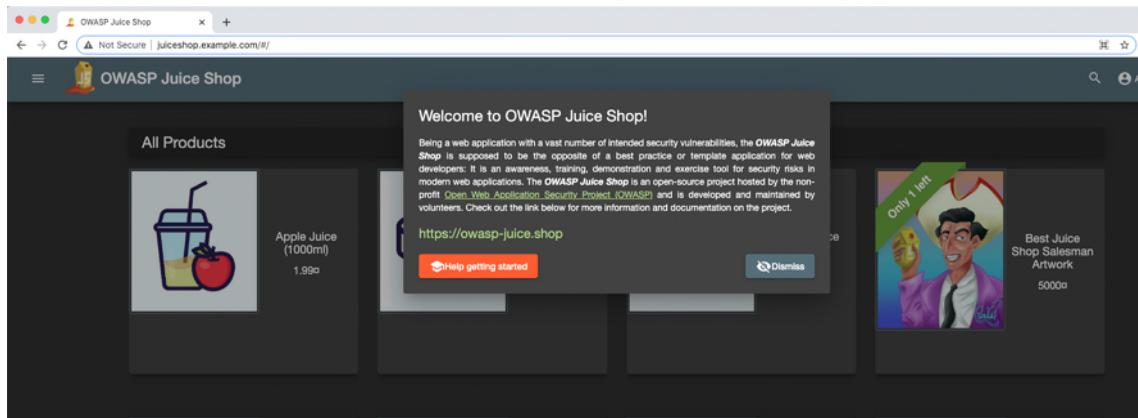
Lab 9: Deploy JuiceShop application, using VS/VSR manifests

In this lab, we deploy a new application, leaving the current Cafe and Bar apps up and running. This new app environment is deployed using NginxPlus VirtualServer/VSRoute manifests.

Learning Objectives

By the end of the lab, you will be able to:

- Create a new Kubernetes namespace
- Deploy and test a new application
- Access this new app thru Nginx Ingress Controller



You will launch a new application, called `Juice Shop`, representing a modern online retail sales app, a nice refreshing addition to Cafe and Bar. It will be deployed in a new Kubernetes namespace called "juice" in your cluster. The Juice Shop app is often used to test various HTTP and Website vulnerabilities. However, you will use it to test various NginxPlus features.

1. Inspect both of the Lab9 Juiceshop YAML files, `juiceshop.yaml`, and `juiceshop-vs.yaml`.
Do you see the deployment, service, and virtual server and route definitions?
2. Next deploy the JuiceShop namespace, application using these manifests:

```
kubectl create namespace juice
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl create namespace juice
namespace/juice created
```

```
kubectl apply -f lab9/juiceshop.yaml
kubectl apply -f lab9/juiceshop-vs.yaml
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab9/juiceshop.yaml
deployment.apps/juiceshop created
service/juiceshop-svc created
secret/juice-secret created
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl apply -f lab9/juiceshop-vs.yaml
virtualserver.k8s.nginx.org/juiceshop-vs created
```

3. Show running Juice components:

```
kubectl get pods,svc,vs -n juice -o wide
```

```
ubuntu@k8-jumphost:~/Documents/nginx-ingress-workshops/labs$ kubectl get pods,svc,vs -n juice -o wide
NAME                               READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
pod/juiceshop-6cf9ddc7f-258w8      1/1    Running   0          95s    192.168.219.18   k8s3   <none>        <none>
pod/juiceshop-6cf9ddc7f-j5g5n      1/1    Running   0          95s    192.168.109.94   k8s2   <none>        <none>
pod/juiceshop-6cf9ddc7f-k65js     1/1    Running   0          95s    192.168.219.15   k8s3   <none>        <none>

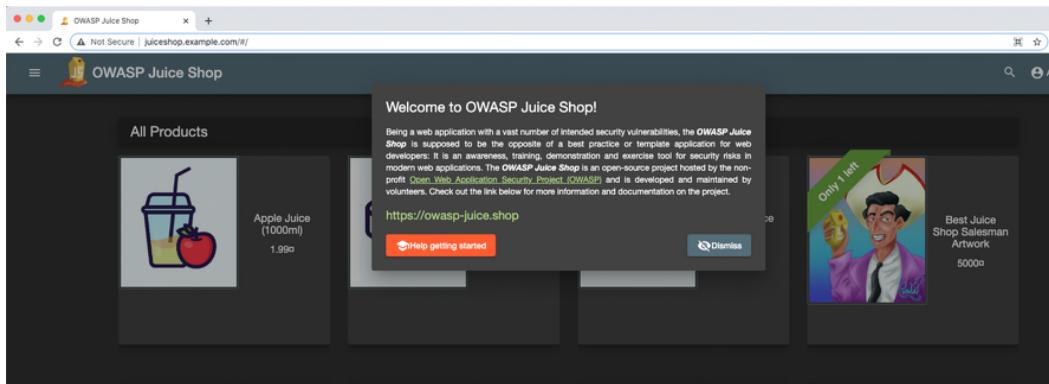
NAME              TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE     SELECTOR
service/juiceshop-svc   ClusterIP  None         <none>       80/TCP    95s    app=juiceshop

NAME                           STATE   HOST          IP          PORTS   AGE
virtualserver.k8s.nginx.org/juiceshop-vs   Valid   juiceshop.example.com   82s
```

4. Test the new Juice Shop application and VS/VSR manifests.

Open Chrome, navigate to (<http://juiceshop.example.com>). Click around for a few minutes to explore the application.

Did you notice, how easy it was with Plus KIC, to launch a completely new application in just a few seconds? With just 2 YAML manifest files - and no IT tickets required to do this? Nginx KIC can perform the Layer7 Hostname and path routing for many different applications running in your k8s cluster.



5. Check the KIC Plus Dashboard, you should see a new JuiceShop HTTP Zone and Upstreams:

Server Zones

Zone	Requests			Responses					Traffic			
	Current	Total	Req/s	Txx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent
dashboard.example.com	2	1364	11	0	1361	0	1	0	1362	20.0 KB	4.31 KB	2.83 MB
bar.example.com	0	0	0	0	0	0	0	0	0	0	0	0
cafe.example.com	0	0	0	0	0	0	0	0	0	0	0	0
juiceshop.example.com	0	207	0	0	92	30	85	0	207	0	0	531 KB
grafana.example.com	0	0	0	0	0	0	0	0	0	0	0	0
prometheus.example.com	0	0	0	0	0	0	0	0	0	0	0	0

HTTP Upstreams

Server	Requests			Responses					Conns		Traffic		Server checks			Health monitors			Response time				
	Name	DT	W	Total	Req/s	Txx	2xx	3xx	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers
192.168.24.186:80	0ms	1	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	14	0	0	passed	-	-
192.168.37.136:80	0ms	1	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	14	0	0	passed	-	-
192.168.9.236:80	0ms	1	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	14	0	0	passed	-	-

vs_juice_juiceshop-vs_juiceshop Zone: 0 %

Server	Requests			Responses					Conns		Traffic		Server checks			Health monitors			Response time	
	Name	DT	W	Total	Req/s	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response	
192.168.13.160:3000	0ms	1	69	0	28	0	0	0	0	41.6 KB	69.8 KB	0	0	14	0	0	passed	4ms	5ms	
192.168.49.121:3000	0ms	1	69	0	28	0	0	0	0	42.2 KB	49.6 KB	0	0	14	0	0	passed	7ms	10ms	
192.168.7.225:3000	0ms	1	69	0	29	0	0	0	0	42.7 KB	19.3 KB	0	0	14	0	0	passed	6ms	6ms	

This completes this Lab.

References:

- [Nginx VirtualServer / Route](#)
- [JuiceShop Demo Source](#)

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
- Shouvik Dutta - Technical Solutions Architect @ F5, Inc.

Navigate to ([Lab10](#) | [Main Menu](#))

Lab 10: Advanced Nginx Plus features with VirtualServer manifests

Learning Objectives

By the end of the lab, you will be able to:

- Enable and test some Nginx Plus features to control how Ingress Controller handles different situations and enables Enterprise features, like:
 - Active Healthchecks
 - Custom Error Pages
 - HTTP Caching
 - Mutual TLS
 - Blue/Green testing

Active Healthchecks



Nginx Plus provides many options for active health checking of pods and services. You will enable some checks for URI paths, and set the interval and counts that meet the needs of the application. A Community Ingress manifest has some limitations in defining a healthcheck. With an Nginx VirtualServer manifest, additional options can be configured for production workloads.

1. Inspect `lab10/juice-health-bad-vs.yaml` file, lines 17-25 for the healthchecks. You will notice we have commented out the correct TCP port for the healthcheck of the backend pods. What do you think will happen if you do not check the correct TCP port?

```
17    healthCheck:
18      enable: true
19      path: /
20      interval: 5s
21      jitter: 3s
22      fails: 3
23      passes: 2
24      #port: 3000
25      connect-timeout: 15s
26      read-timeout: 5s
27      statusMatch: "200"
```

2. Remove the running JuiceShop Virtual Server from the previous lab:

```
kubectl delete -f lab9/juiceshop-vs.yaml
```

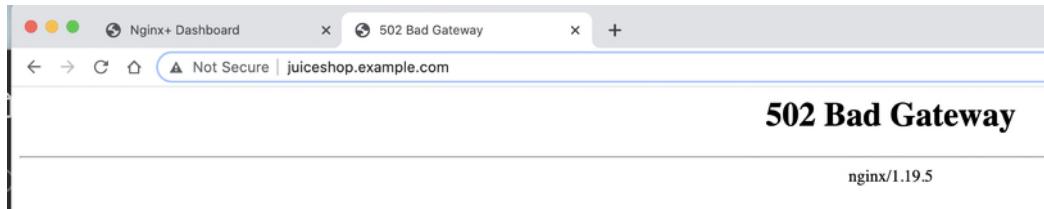
3. Try the new Virtual Server with incorrect healthchecks enabled:

```
kubectl apply -f lab10/juice-health-bad-vs.yaml
```

4. Check your Plus Dashboard, and then try refreshing your Juice Shop browser page. What did you see and what happened to your JuiceShop application ?

► Click for Hints!

vs_juice_juiceshop-vs_juiceshop Zone: 10 %																	
Server	Name	Requests		Responses		Conns		Traffic		Server checks		Health monitors					
		Total	Req/s	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last
10.244.0.205:3000		14.6s	1	0	0	0	0	0	0	0	0	0	0	5	5	1	Failed
10.244.0.25:3000		13.24s	1	0	0	0	0	0	0	0	0	0	0	5	5	1	Failed
10.244.1.42:3000		12.19s	1	0	0	0	0	0	0	0	0	0	0	5	5	1	Failed



5. Quickly!! - checking further, look at the Ingress Controller logs, what do they show ?

```
kubectl logs -n nginx-ingress $KIC_POD_NAME --follow --tail=20
```

Detailed Explanation: The Dashboard shows all your juiceshop upstreams as Down, due to Failed healthchecks. Your JuiceShop VirtualServer is running, but the website is now offline because the pods are in a Failed state. Nginx 502 Bad Gateway errors are an important sign that Nginx has no upstreams available to service the request.

The Ingress Logs should show [error] 97 ... 111: Connection refused messages for the healthchecks on port 80 (but remember, the JuiceShop pods are running on port 3000!).

```
NFD-ML-00026757:nginx-ingress-workshops akker$ kubectl logs -n nginx-ingress nginx-ingress-796ccc4bdf-kg6zk --follow --tail=20
2021/06/30 22:30:16 [error] 97#97: connect() failed (111: Connection refused) while connecting to upstream, health check "vs_juice_juiceshop-vs_juiceshop_match" of peer 10.244.0.205:3000 in upstream "vs_juice_juiceshop-vs_juiceshop" port 80
2021/06/30 22:30:20 [error] 97#97: connect() failed (111: Connection refused) while connecting to upstream, health check "vs_juice_juiceshop-vs_juiceshop_match" of peer 10.244.0.25:3000 in upstream "vs_juice_juiceshop-vs_juiceshop" port 80
2021/06/30 22:30:22 [error] 97#97: connect() failed (111: Connection refused) while connecting to upstream, health check "vs_juice_juiceshop-vs_juiceshop_match" of peer 10.244.0.205:3000 in upstream "vs_juice_juiceshop-vs_juiceshop" port 80
2021/06/30 22:30:24 [error] 97#97: connect() failed (111: Connection refused) while connecting to upstream, health check "vs_juice_juiceshop-vs_juiceshop_match" of peer 10.244.1.42:3000 in upstream "vs_juice_juiceshop-vs_juiceshop" port 80
```

Type Ctrl+C to stop the log tail when you are finished.

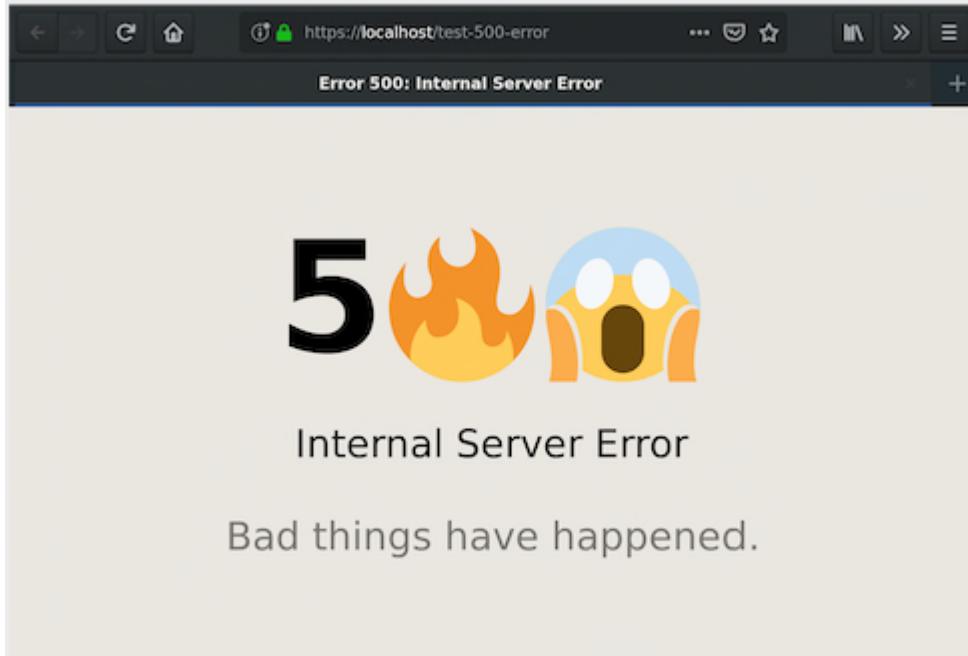
6. Inspect the fixed YAML manifest VS file, juice-health-good-vs.yaml , with the correct healthcheck port of 3000 on line #24. Now try that one:

```
kubectl apply -f lab10/juice-health-good-vs.yaml
```

7. Check your KIC Plus Dashboard again, with Health monitors now green, and your website is up and browser access is restored. The connection errors in the Ingress log should have stopped as well.

vs.juice.juiceshop-vs.juiceshop Zone: 0 %															
Server	Name	DT	W	Requests		Responses		Conns		Traffic		Server checks			
				Total	Req/s	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail
	10.244.0.205:3000	0ms	1	0	0	0	0	=	=	0	0	0	0	0	0
	10.244.0.25:3000	0ms	1	0	0	0	0	=	=	0	0	0	0	0	0
	10.244.1.42:3000	0ms	1	0	0	0	0	=	=	0	0	0	0	0	0

Custom Error Pages



The Director of Customer Support has asked if you can stop the ugly HTTP 502 error messages from going back to the customers, as the developers say they are too busy to fix it. While you can't actually stop them, you can hide the 502 errors and send the customers an alternative page.

So you will enable a `Sorry` page that gives customers a more friendly `Please try again later` message, with a Customer Support phone number to call if they need help.

Nginx Plus provides many options for intercepting HTTP response errors and providing user-friendly error pages from web applications. In this example, you will enable a simple error response page.

1. Inspect `lab10/juice-sorrypage.yaml` file, lines 30-40.

```
27   routes:
28     - path: /
29       action:
30         pass: juiceshop
31       errorPages:
32         - codes: [502]
33           return:
34             code: 200
35             type: application/json
36             body: |
37               {"We apologize for the inconvenience. Please try again in a few minutes, or call 1-888-JUICESHOP."}
38           headers:
39             - name: X-Debug-Original-Status
40               value: ${upstream_status}
```

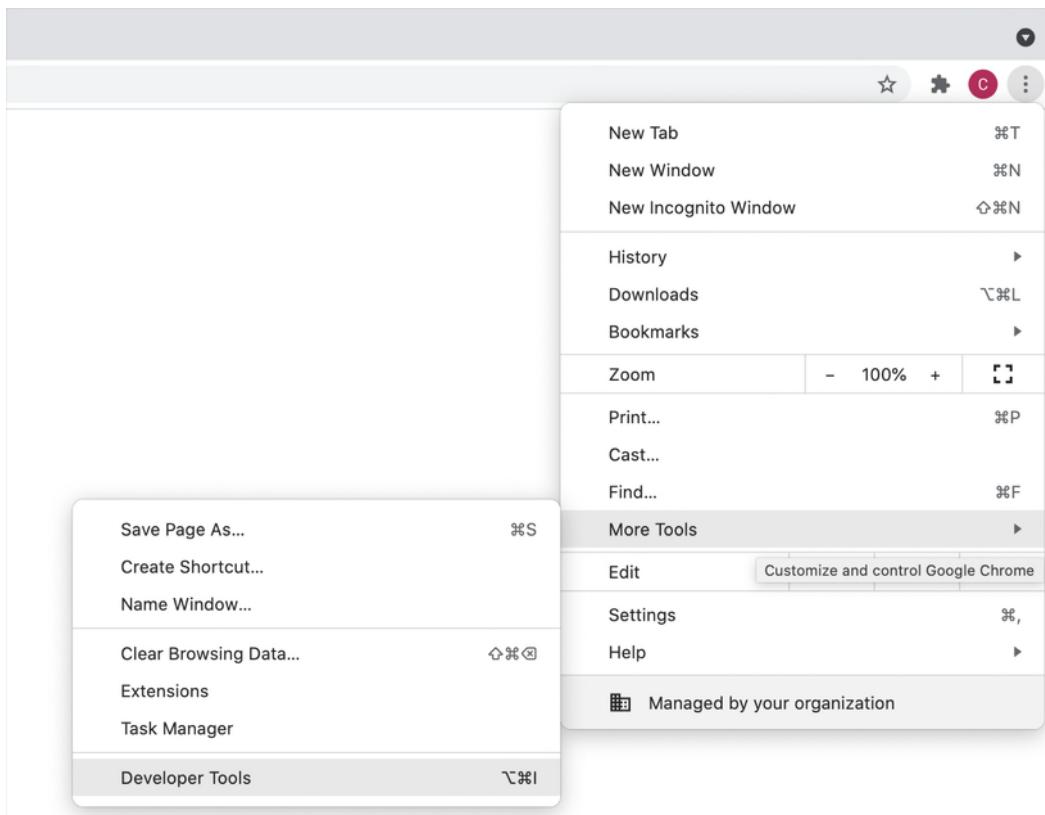
Now apply the customer friendly error page manifest:

```
kubectl apply -f lab10/juice-sorrypage.yaml
```

2. And try a refresh on your Juice Shop Browser, what do you see now ?



Notice we have modified this 502 Bad Gateway error page to be more customer friendly. For production, you could further customize this page if a user encountered an error page from a pod. If you check Chrome Developer Tools, you will see that we have also added a custom Nginx Debug Header , which shows what the original response error code was.

A screenshot of the Chrome Developer Tools Network tab. The tab is selected and shows a list of network requests. One request from 'juiceshop.example.com' is highlighted. The details panel on the right shows the following information for this request:

- Request URL: <https://juiceshop.example.com/>
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 68.183.250.246:443
- Referrer Policy: strict-origin-when-cross-origin
- Response Headers:
 - Connection: keep-alive
 - Content-Length: 101
 - Content-Type: application/json
 - Date: Wed, 30 Jun 2021 23:02:51 GMT
 - Server: nginx/1.19.5
 - X-Debug-Original-Status: 502

Leave the Chrome Developer Tools open, you will need it for the next section.

Nginx Caching



Next, you will use some of the extra RAM available in your Ingress Controller to provide caching of static images from the pods. This will improve the customer experience by delivering images from the KIC's RAM, instead of waiting for the pods to deliver them.

In the previous Enhanced Logging lab, you added the `cache status` variable - `$upstream_cache_status` - to the Nginx access log, so you can see the cache HITS, MISSES, and EXPIRED status in the access log. You will also insert a custom HTTP Header for X-Cache-Status, so we can see the Nginx cache Response Header values with Chrome Developer Tools.

Inspect `lab10/juice-cache-vs.yaml` file, lines 7-9. Notice you are using an `http-snippet` to customize Nginx to use 256MB of available RAM for the cache, and to add an Nginx `X-Cache-Status` HTTP response header. And, on lines 32-36, you are also using a `location-snippet` to further customize Nginx to cache 200 responses for 30 seconds (cache aging timer), and ignore any Cache-Control request headers.

HTTP Snippet	Location Snippet
	<pre>lab10 > ! juice-cache-vs.yaml 31 - path: ~*\.\.css js jpg png 32 location-snippets: 33 proxy_cache juiceshop; 34 proxy_cache_valid 200 30s; 35 proxy_cache_methods GET; 36 proxy_ignore_headers Cache-Control; 37 action: 38 pass: juiceshop</pre>

```

lab10 > ! juice-cache-vs.yaml
1  apiVersion: k8s.nginx.org/v1
2  kind: VirtualServer
3  metadata:
4    name: juiceshop-vs
5    namespace: juice
6  spec:
7    http-snippets: |
8      proxy_cache_path /tmp levels=1:2
9        keys_zone=juiceshop:10m
10       max_size=256m;
11       add_header X-Cache-Status
12       $upstream_cache_status;

```

Note: In production, you would like to set the cache aging timer higher than 30 seconds. We intentionally set the timer low in this lab so you can see what happens when the cache timer expires.

1. Now apply this Caching configuration, and test it:

```
kubectl apply -f lab10/juice-cache-vs.yaml
```

Monitor the Ingress Controller access log, watch for "HIT", "MISS", and "EXPIRED" entries, while you refresh the Juice Shop pages:

```
kubectl logs -n nginx-ingress $KIC_POD_NAME --follow --tail 50
```

During refreshes, you should see some Cache "MISS" and "HIT" and "EXPIRED" log entries, it should be the last field of each log entry in the access log, as shown below. (This Cache Status logging variable was added when you enabled Enhanced Logging in a previous exercise).

```

10.244.0.215 -- [01/Jul/2021:00:12:59 +0000] "GET /assets/public/images/products/green_smoothie.jpg HTTP/1.1" 200 15910 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "bf50b059b6bb7cfed347b0b022ed4d6" "0.002" "10.244.0.25:3000" "0.000" "0.000" "0.000" "0.000" "MISS"
10.244.1.85 -- [01/Jul/2021:00:12:59 +0000] "GET /assets/public/images/products/lemon_juice.jpg HTTP/1.1" 200 17038 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "30560e6e3d78a4b6e9fedc13efb500f5" "0.012" "10.244.0.25:3000" "0.000" "0.012" "0.012" "0.000" "MISS"
10.244.0.215 -- [01/Jul/2021:00:12:59 +0000] "GET /assets/public/images/products/permafrost.jpg HTTP/1.1" 200 101076 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "846cb9c7e11299a2d17a3086bec43d" "0.012" "10.244.0.25:3000" "0.000" "0.004" "0.012" "0.000" "MISS"
138.68.45.130 -- [01/Jul/2021:00:12:59 +0000] "GET /assets/public/images/products/fan_facemask.jpg HTTP/1.1" 200 26934 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "70c88f8fdbb8c64d59010a434eb2c18" "0.011" "10.244.0.25:3000" "0.000" "0.008" "0.008" "0.000" "MISS"
NFD-ML-00026757:nginx-ingress-workshop$ akers

```

```

138.68.45.130 -- [06/Jul/2021:17:15:39 +0000] "GET /assets/public/images/products/lemon_juice.jpg HTTP/1.1" 200 17038 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "c1d31e329334ce230ad15e5dba27d883" "0.000" "--" "--" "--" "--" "HIT"
10.244.1.85 -- [06/Jul/2021:17:15:39 +0000] "GET /assets/public/images/products/green_smoothie.jpg HTTP/1.1" 200 15910 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "f16c4a028b9fe152f5f6c3f8d71eb09" "0.000" "--" "--" "--" "--" "HIT"
10.244.0.215 -- [06/Jul/2021:17:15:39 +0000] "GET /assets/public/images/products/permafrost.jpg HTTP/1.1" 200 101076 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "f71fa07622b0812ec0ecb33052a37945" "0.000" "--" "--" "--" "--" "HIT"
138.68.45.130 -- [06/Jul/2021:17:15:39 +0000] "GET /assets/public/images/products/fan_facemask.jpg HTTP/1.1" 200 26934 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "d916a18d3956b0ae398a1586cc07f3cc" "0.000" "--" "--" "--" "--" "HIT"

```

```

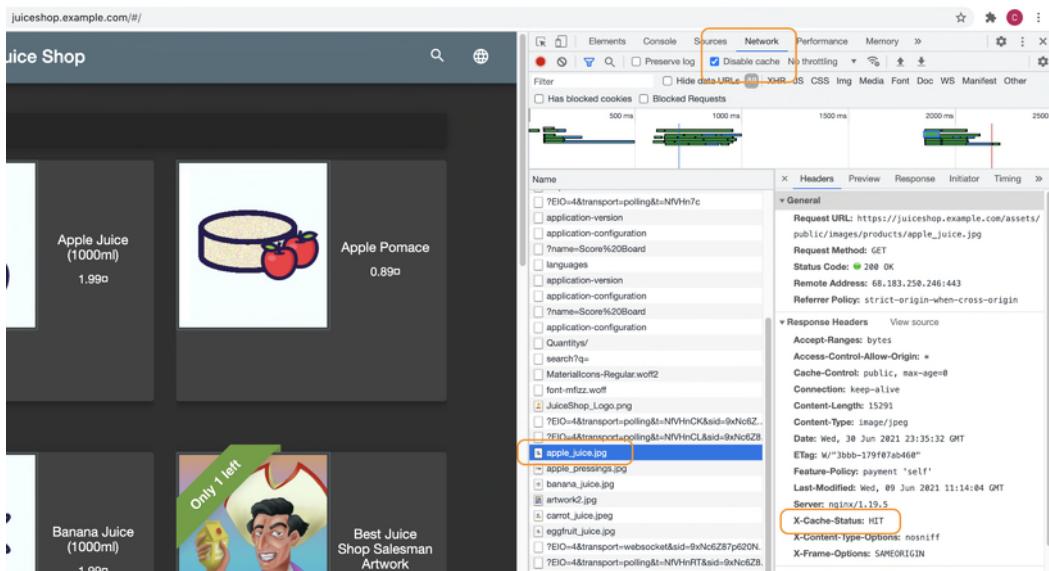
138.68.45.130 - - [30/Jun/2021:23:30:53 +0000] "GET /assets/public/images/products/banana_juice.jpg HTTP/1.1" 200 19833 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "e76032915e25c398b747f03730afbb1" "0.013" "10.244.1.42:3000" "0.000" "0.000" "0.012" "0.012" "0.000" "EXPIRED"
10.244.1.85 - - [30/Jun/2021:23:30:53 +0000] "GET /assets/public/images/products/apple_pressings.jpg HTTP/1.1" 200 29163 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "6739b69ca15d91e5a8f02e67afbf60" "0.012" "10.244.1.42:3000" "0.000" "0.004" "0.012" "0.012" "0.000" "EXPIRED"
10.244.0.215 - - [30/Jun/2021:23:30:53 +0000] "GET /assets/public/images/products/apple_juice.jpg HTTP/1.1" 200 15291 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "2baab91f72caf94a601a7a0e0069ba80" "0.012" "10.244.1.42:3000" "0.000" "0.004" "0.020" "0.020" "0.000" "EXPIRED"
10.244.0.215 - - [30/Jun/2021:23:30:53 +0000] "GET /assets/public/images/products/artwork2.jpg HTTP/1.1" 200 37081 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "b549aae07fcbeb00bc56289c2b8d51d1" "0.020" "10.244.0.25:3000" "0.000" "0.004" "0.020" "0.020" "0.000" "EXPIRED"
10.244.0.215 - - [30/Jun/2021:23:30:53 +0000] "GET /assets/public/images/products/green_smoothie.jpg HTTP/1.1" 200 15010 "https://juiceshop.example.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36" "-" "juiceshop-vs" "virtualserver" "juice" "juiceshop-svc" "5a0a7a0e0069ba80" "0.012" "10.244.1.42:3000" "0.000" "0.004" "0.012" "0.012" "0.000" "EXPIRED"

```

Type Ctrl+C to stop the log tail when you are finished.

- Now, open a new Tab in Chrome, enable Developer Tools, and make sure you disable Chrome's internal browser cache, and select the Network tab at the top in Chrome Tools. Click the JuiceShop Favorite.

Look for the X-Cache-Status Response Header and value (that Ingress is adding). It should look like this:



- Deep Dive** - As an example, if you click on the first image in the Name list, `apple_juice.jpg`, you should see an HTTP Response Header `X-Cache-Status = MISS`. If you refresh a couple times, and check again, it should now show `X-Cache-Status = HIT`. If you wait more than 30 seconds, and refresh again, it should show `X-Cache-Status = EXPIRED`.

Explanation: the first request will be a cache MISS, because Nginx does not have a copy of this object in the Cache, and it must be served from the Upstream origin pod. After the first request, it will be a Cache HIT because Nginx is caching it and served it from its Cache. After the age timer expires, you will see EXPIRED.

This is because the Nginx `proxy_cache_valid` directive is set to 30 seconds, on line #32 of the manifest YAML file.

Screenshot of the Network tab in the Chrome DevTools developer console.

The Network tab shows a timeline of network requests. A red vertical line marks the start of the selected request for "apple_juice.jpg".

Request URL: https://juiceshop.example.com/assets/public/images/products/apple_juice.jpg

Request Method: GET

Status Code: 200 OK

Remote Address: 68.183.250.246:443

Referrer Policy: strict-origin-when-cross-origin

Response Headers:

- Accept-Ranges: bytes
- Access-Control-Allow-Origin: *
- Cache-Control: public, max-age=0
- Connection: keep-alive
- Content-Length: 15291
- Content-Type: image/jpeg
- Date: Wed, 30 Jun 2021 23:47:45 GMT
- ETag: W/"3bbb-179f07ab460"
- Feature-Policy: payment 'self'
- Last-Modified: Wed, 09 Jun 2021 11:14:04 GMT
- Server: nginx/1.19.5
- X-Cache-Status: MISS
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN

Name

- languages
- application-version
- application-configuration
- ?name=Score%20Board
- application-configuration
- Quantitys/
- search?q=
- MaterialIcons-Regular.woff2
- font-mfizz.woff
- JuiceShop_Logo.png
- ?EIO=4&transport=polling&t=NfVKaC1&sid=T0w4T6..
- ?EIO=4&transport=polling&t=NfVKaC3&sid=T0w4T6..
- apple_juice.jpg**
- apple_pressings.jpg
- banana_juice.jpg
- artwork2.jpg
- carrot_juice.jpeg
- eggfruit_juice.jpg
- ?EIO=4&transport=websocket&sid=T0w4T6S2bHOA..
- ?EIO=4&transport=polling&t=NfVKaQt&sid=T0w4T6..
- fruit_press.jpg
- green_smoothie.jpg
- permafrost.jpg
- lemon_juice.jpg

Screenshot of the Network tab in the Chrome DevTools developer tools interface.

Network Tab Options:

- Elements
- Console
- Sources
- Network** (selected)
- Performance
- Memory
- 6 1
- Preserve log
- Disable cache (checked)
- No throttling
- Filter: All XHR JS CSS Img Media Font Doc WS Manifest Other
- Has blocked cookies
- Blocked Requests

Request Details for "apple_juice.jpg":

General:

- Request URL: https://juiceshop.example.com/assets/public/images/products/apple_juice.jpg
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 68.183.250.246:443
- Referrer Policy: strict-origin-when-cross-origin

Response Headers:

- Accept-Ranges: bytes
- Access-Control-Allow-Origin: *
- Cache-Control: public, max-age=0
- Connection: keep-alive
- Content-Length: 15291
- Content-Type: image/jpeg
- Date: Tue, 06 Jul 2021 17:15:39 GMT
- ETag: W/"3bbb-179f07ab460"
- Feature-Policy: payment 'self'
- Last-Modified: Wed, 09 Jun 2021 11:14:04 GMT
- Server: nginx/1.19.5
- X-Cache-Status: HIT
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN

Request Headers:

- Accept: image/avif, image/webp, image/apng, image/svg+xml, image/*,*/*;q=0.8
- Accept-Encoding: gzip, deflate, br
- Accept-Language: en-US,en;q=0.9
- Cache-Control: no-cache
- Connection: keep-alive
- Cookie: language=en; welcomebanner_status=dismiss;

Resource List:

- Quantity/
- search?q=
- MaterialIcons-Regular.woff2
- font-mfizz.woff
- JuiceShop_Logo.png
- ?EIO=4&transport=polling&t=NfyqN4x&sid=2qOUbD..
- ?EIO=4&transport=polling&t=NfyqN4z&sid=2qOUbDy
- apple_juice.jpg** (Selected)
- apple_pressings.jpg
- banana_juice.jpg
- artwork2.jpg
- carrot_juice.jpeg
- eggfruit_juice.jpg
- ?EIO=4&transport=websocket&sid=2qOUbDyEhSBcl..
- ?EIO=4&transport=polling&t=NfyqNJC&sid=2qOUbD..
- fruit_press.jpg
- green_smoothie.jpg
- permafrost.jpg
- lemon_juice.jpg
- melon_bike.jpeg
- fan_facemask.jpg
- ?EIO=4&transport=polling&t=NfyqNKc&sid=2qOUbD..
- favicon_js.ico
- ?EIO=4&transport=polling&t=NfyqNg7
- ?EIO=4&transport=polling&t=NfyqNgm&sid=z1QG5tU..
- ?EIO=4&transport=polling&t=NfyqNgo&sid=z1QG5tU..
- ?EIO=4&transport=polling&t=NfyqNhO&sid=z1QG5tU..
- ?EIO=4&transport=polling&t=NfyqNzX
- ?EIO=4&transport=polling&t=NfyqN-8&sid=ShQqhRh..
- ?EIO=4&transport=polling&t=NfyqN-A&sid=ShQqhRh..
- ?EIO=4&transport=polling&t=NfyqN-p&sid=ShQqhRh..

53 requests | 1.1 MB transferred | 3.2 MB resources

The screenshot shows the Chrome Developer Tools Network tab. A request for `apple_juice.jpg` is selected. The Headers section shows the following response headers:

- Request URL: `https://juiceshop.example.com/assets/public/images/products/apple_juice.jpg`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 68.183.250.246:443
- Referrer Policy: strict-origin-when-cross-origin
- Accept-Ranges: bytes
- Access-Control-Allow-Origin: *
- Cache-Control: public, max-age=0
- Connection: keep-alive
- Content-Length: 15291
- Content-Type: image/jpeg
- Date: Wed, 30 Jun 2021 23:49:24 GMT
- ETag: W/"3bbb-179f07ab460"
- Feature-Policy: payment 'self'
- Last-Modified: Wed, 09 Jun 2021 11:14:04 GMT
- Server: nginx/1.19.5
- X-Cache-Status: EXPIRED
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN

4. Check the Plus Dashboard, did you find the Cache statistics ? You see some basic stats about Nginx Caching, the memory and disk usage, Hit Ratio, and Bytes served, changing in real-time as you browse around the Juice Shop site.

How high can you get your hit ratio ?

Zone	State	Memory usage	Max size	Used	Disk usage	Traffic	Hit Ratio
Juiceshop	!	1 %	256 MB	928 KB	0 %	12.8 MB 11.1 MB 11.1 MB	<div style="width: 60%;">60%</div>

5. Again using Chrome Developer tools, look at the `carrot_juice.jpeg` object. What is different ? Why ?

The screenshot shows the Chrome DevTools Network tab. The timeline at the top indicates a total duration of 30 seconds. Below the timeline, a table lists network requests. The row for 'carrot_juice.jpeg' is highlighted with a blue selection bar and has its details expanded.

Name	Headers	Preview	Response	Initiator	Timing
?name=Score%20Board					
languages					
application-version					
application-configuration					
?name=Score%20Board					
application-configuration					
Quantitys/					
search?q=					
MaterialIcons-Regular.woff2					
font-mfizz.woff					
JuiceShop_Logo.png					
?EIO=4&transport=polling&t=NfVKyPQ&sid=TBWEpV.					
?EIO=4&transport=polling&t=NfVKyPS&sid=TBWEpV.					
?EIO=4&transport=polling&t=NfVKyRw&sid=TBWEpV.					
favicon_js.ico					
apple_juice.jpg					
apple_pressings.jpg					
banana_juice.jpg					
artwork2.jpg					
carrot_juice.jpeg					
eggfruit_juice.jpg					
fruit_press.jpg					
green_smoothie.jpg					
permafrost.jpg					

General

Request URL: https://juiceshop.example.com/assets/public/images/products/carrot_juice.jpeg
Request Method: GET
Status Code: 200 OK
Remote Address: 68.183.250.246:443
Referrer Policy: strict-origin-when-cross-origin

Response Headers

Accept-Ranges: bytes
Access-Control-Allow-Origin: *
Cache-Control: public, max-age=0
Connection: keep-alive
Content-Length: 19001
Content-Type: image/jpeg
Date: Wed, 30 Jun 2021 23:49:24 GMT
ETag: W/"4a39-179f07ab460"
Feature-Policy: payment 'self'
Last-Modified: Wed, 09 Jun 2021 11:14:04 GMT
Server: nginx/1.19.5
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN

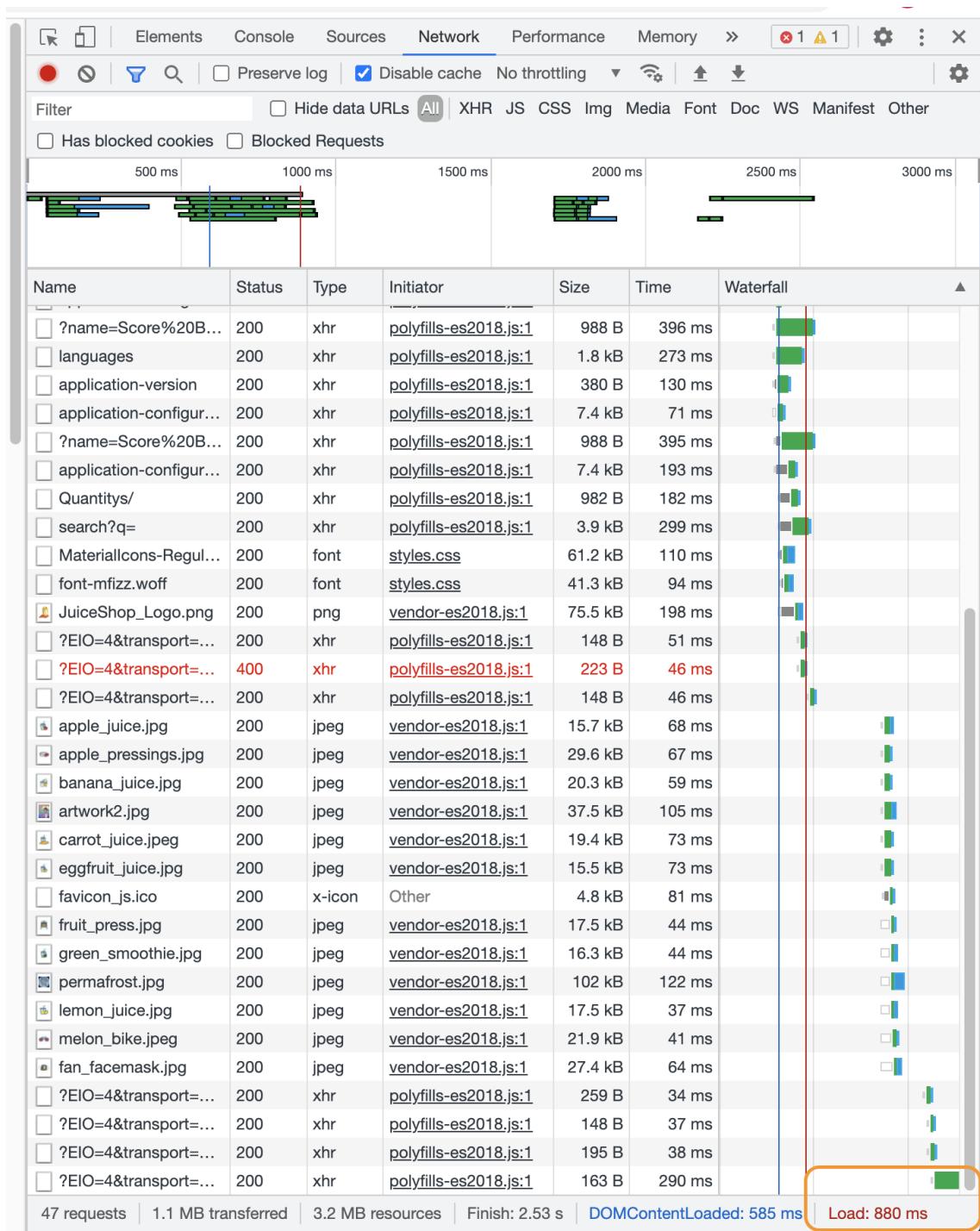
Request Headers

► Click for Hints!

```
lab10 > ! juice-cache-vs.yaml
```

```
31     - path: ~*\*.css|js|jpg|png
32         location-snippets: |
33             proxy_cache juiceshop;
34             proxy_cache_valid 200 30s;
35             proxy_cache_methods GET;
36             proxy_ignore_headers
            Cache-Control;
```

Inspect the very bottom right corner of Chrome Developer Tools...what is the difference in Page Load times, when there are Nginx Cache Hits, vs Cache Misses (refresh the page 5 or 6 times, then wait more than 30 seconds and refresh again)? You should see the Page Load time much faster with Cache Hits, of course. How fast could you get your page load time?



Final Check - did you notice just how much difference in response time there is, between the different JuiceShop pods ?

It's important to have Advanced features like LeastTime LastByte load balancing, and Caching , to help improve the user experience with imbalanced pods that may have poor performance.

vs_juice_juiceshop-vs_juiceshop Zone: 0 %														Show all ▾							
Server	Name	Requests		Responses		Conns		Traffic			Server checks		Health monitors			Response time					
		DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers
10.244.0.205:3000	0ms	1	476	0		23	0	0	=	0	0	388 kB	5.89 MiB	0	0	167	0	0	passed	37ms	24308ms
10.244.0.25:3000	0ms	1	355	0		31	0	0	=	0	0	293 kB	4.41 MiB	0	0	168	0	0	passed	129ms	26716ms
10.244.1.42:3000	0ms	1	555	0		23	0	1	=	0	0	457 kB	8.75 MiB	0	0	167	0	0	passed	89ms	171ms

Mutual TLS



The Boss's business insurance auditor has informed him that his website must have SSL encryption for all his web traffic from end-to-end. Not just the traffic between his customers and the Ingress Controller, but also between the Ingress Controller and all of the application pods. This is called mutualTLS for the Ingress Controller. So now you have to configure/enable TLS between Ingress Controller and the Cafe coffee and tea pods, and verify that it works.

Inspect the `lab10/cafe-mtls.yaml`, lines 19, and 29-30. Notice the change from port 80 to port 443 for the container and the Service, to use for mutual TLS.

Container	Service
	<pre>lab10 > ! cafe-mtls.yaml 20 --- 21 apiVersion: v1 22 kind: Service 23 metadata: 24 name: coffee-mtls-svc 25 spec: 26 type: ClusterIP 27 clusterIP: None 28 ports: 29 - port: 443 30 targetPort: 443 31 protocol: TCP 32 name: https</pre>

```
lab10 > ! cafe-mtls.yaml
 1  apiVersion: apps/v1
 2  kind: Deployment
 3  metadata:
 4    name: coffee-mtls
 5  spec:
 6    replicas: 3
 7    selector:
 8      matchLabels:
 9        app: coffee-mtls
10    template:
11      metadata:
12        labels:
13          app: coffee-mtls
14      spec:
15        containers:
16          - name: coffee-mtls
17            image: nginxinc/ingress-demo
18          ports:
19            - containerPort: 443
20 ---
```

1. First, for proper testing, you need to remove the previous Cafe virtual server, as we will still be using the `cafe.example.com` hostname:

```
kubectl delete vs cafe-vs
```

2. Check the Plus Dashboard, the Cafe HTTP Zone should now be gone.
3. Start a fresh Cafe Demo, deploy the mTLS enabled pods and services and Virtual Server manifests:

```
kubectl apply -f lab10/cafe-mtls.yaml
kubectl apply -f lab10/cafe-mtls-vs.yaml
```

Question - Will changing the pods from port 80 to port 443 break the previously defined Healthchecks?

► Click for Hints!

4. Check the Plus Dashboard, and your new mTLS Cafe Application - ensure all 6 "mtls" coffee and tea pods are now in Up/Green status.

vs_default [cafe-mtls-vs_coffee-mtls] Zone: 0 % Show all ▾

Server	Requests			Responses			Conns		Traffic			Server checks		Health monitors			Response time			
	DT	W	Total	Req/s	-	4xx	Sxx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers
10.244.1.1:443	0ms	1	0	0	=	0	0	0	0	0	0	0	0	0	5	0	0	passed	-	-
10.244.0.249:443	0ms	1	0	0	►	0	0	0	0	0	0	0	0	0	5	0	0	passed	-	-
10.244.0.36:443	0ms	1	0	0	=	0	0	0	0	0	0	0	0	0	5	0	0	passed	-	-

vs_default [cafe-mtls-vs_tea-mtls] Zone: 0 % Show all ▾

Server	Requests			Responses			Conns		Traffic			Server checks		Health monitors			Response time			
	DT	W	Total	Req/s	-	4xx	Sxx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers
10.244.1.11:443	0ms	1	0	0	=	0	0	0	0	0	0	0	0	0	3	0	0	passed	-	-
10.244.0.219:443	0ms	1	0	0	►	0	0	0	0	0	0	0	0	0	3	0	0	passed	-	-
10.244.0.102:443	0ms	1	0	0	=	0	0	0	0	0	0	0	0	0	3	0	0	passed	-	-

5. Using Chrome, check the access to coffee and tea as before:

<https://cafe.example.com/coffee>

<https://cafe.example.com/tea>

Do you see the pod Server Name now shows coffee-mtls-pod-name and tea-mtls-pod-name ?

Do you see the pod Server Address now shows port 443, and not 80 ?



Server Name:	coffee-mls-678c888d87-cdkbm
Server Address:	10.244.1.2:443
NGINX Version:	1.19.6
Request Date:	06/Jul/2021:18:09:09 +0000
Request URI:	/coffee
Document Root:	/usr/share/nginx/html/beverage
Ingress Controller IP:	10.244.0.100:56092
Client IP:	10.244.1.85

Auto Refresh

Request ID: 53ea41f2282ecba2f712b52e292cd3d4
© NGINX, Inc. 2020

Blue/Green | A/B Testing



During the development cycle of modern applications for Kubernetes, developers will often want to test new versions of their software, using various test tools, and ideally, a final check with live customer traffic. There are several names for this dev/test concept - Blue/Green deployments, A/B testing, Canary testing, etc.

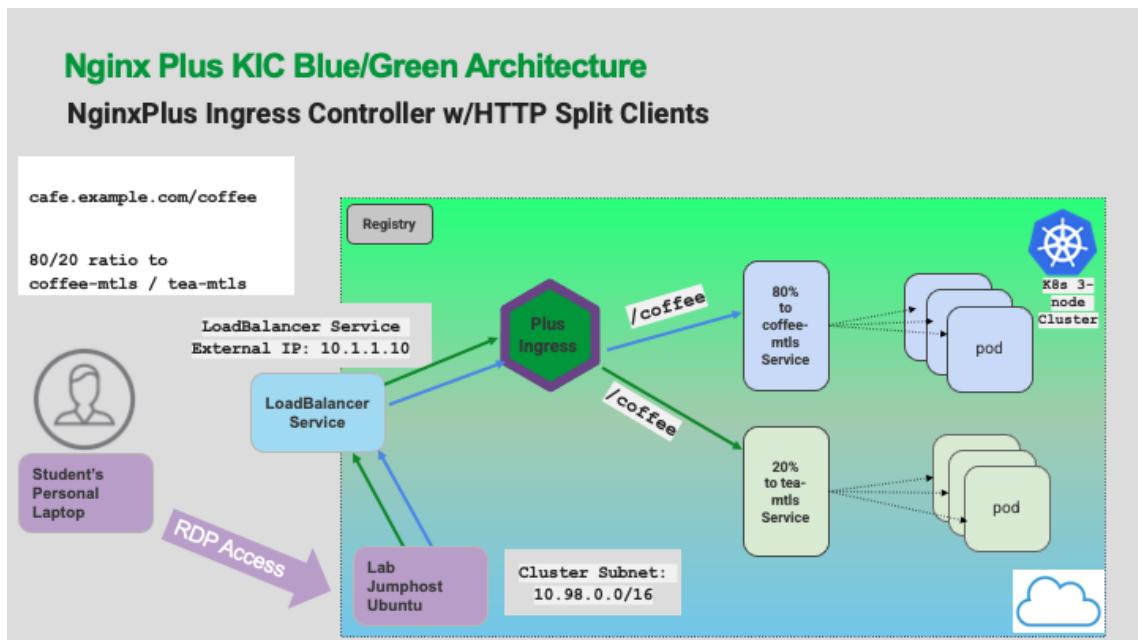
However, switching ALL customers to new versions that might still have a few bugs in the code is quite risky.

Wouldn't it be nice if your Ingress Controller could split off just a small fraction of your live traffic, and route it to your new application pod for final testing?

Nginx Plus Ingress Controller can do this, using a feature called HTTP Split Clients. This feature allows you to define a percentage of traffic to be split between different k8s Services, representing different versions of your application.

You will use the currently running Cafe-mTLS coffee and tea pods, and split the traffic at an 80:20 ratio between coffee and tea Services.

Refer to the following diagram for testing Blue/Green traffic splitting with Nginx Plus Ingress Controller:



Assume that coffee is your existing application, and tea is your new test build of the application.

Having read the tea leaves you are highly confident in your new code. So you decide to route 20% of your live traffic to tea. (crossing your fingers)

1. First, to see the split ratio more clearly, scale down the number of coffee and tea pods to just one each:

```
kubectl scale deployment coffee-mtls --replicas=1  
kubectl scale deployment tea-mtls --replicas=1
```

2. Check the Plus Dashboard, there should only be one coffee and one tea upstream now.

The screenshot shows two VirtualServer configurations in the Istio Plus Dashboard:

vs_default_cafe-mtls-vs_coffee-mtls Zone: 12 %

Server	DT	W	Total	Req/s	Responses	Conns	Traffic
Name 10.244.1.2:443	0ms	1	2	0	0xx 0 0xx 0 0 0 0 0	A L	Sent/s Rcv

vs_default_cafe-mtls-vs_tea-mtls Zone: 10 %

Server	DT	W	Total	Req/s	Responses	Conns	Traffic
Name 10.244.1.117:443	0ms	1	0	0	0xx 0 0xx 0 0 0 0 0	A L	Sent/s

Inspect the `lab10/cafe-bluegreen-vs.yaml` file, and note the `split` and `weight` directives on lines 49-56.

```
lab10 > ! cafe-bluegreen-vs.yaml
49   - path: /coffee
50     splits:
51       - weight: 80
52         action:
53           |   pass: coffee-mtls
54       - weight: 20
55         action:
56           |   pass: tea-mtls
57
58
```

3. Next, remove the existing VirtualServer for mTLS from the previous exercise:

```
kubectl delete -f lab10/cafe-mtls-vs.yaml
```

4. Now configure the Cafe Virtual Server to send 80% traffic to coffee-mtls, and 20% traffic to tea-mtls:

```
kubectl apply -f lab10/cafe-bluegreen-vs.yaml
```

5. Open a Chrome tab for <https://cafe.example.com/coffee>, and check the Auto Refresh box at the bottom of the page.



Server Name:	coffee-mtls-678c888d87-cdkbm
Server Address:	10.244.1.2:443
NGINX Version:	1.19.6
Request Date:	06/Jul/2021:18:41:47 +0000
Request URI:	/coffee
Document Root:	/usr/share/nginx/html/beverage
Ingress Controller IP:	10.244.0.100:41080
Client IP:	10.244.1.85

Auto Refresh

Request ID: d794c70563224073e81edceceb50e729
© NGINX, Inc. 2020

Check the statistics on the Plus Dashboard cafe-bluegreen upstreams.... Do you see approximately an 80/20 Requests ratio between coffee and tea? You can configure the ratio in 1% increments, from 1-99%.

Note: Nginx will not load the Split configuration, if the ratio does not add up to 100%.

vs_default_cafe-bluegreen-vs_coffee-mtls								Zone: 12 %		
Server	Name	DT	W	Requests		Responses		Conns	T	
				Total	Req/s	...	4xx	5xx	A	L
10.244.1.2:443		0ms	1	804	0	▶	0	0	0	∞

vs_default_cafe-bluegreen-vs_tea-mtls								Zone: 12 %		
Server	Name	DT	W	Requests		Responses		Conns	T	
				Total	Req/s	...	4xx	5xx	A	L
10.244.1.117:443		0ms	1	194	0	▶	0	0	0	∞

Important! You are still using the <https://cafe.example.com/coffee> URL - you did not have to change the PATH of the url, but Nginx Ingress Controller is routing the requests to 2 different services, 80% to coffee-mtls AND 20% to tea-mtls! This allows for easy testing of new application versions, without requiring DNS changes, new URLs or URIs, or other system changes.

References

- Active Healthchecks: <https://docs.nginx.com/nginx-ingress-controller/configuration/virtualserver-and-virtualserverroute-resources/#upstream-healthcheck>
- Error Pages: <https://docs.nginx.com/nginx-ingress-controller/configuration/virtualserver-and-virtualserverroute-resources/#errorpage>
- Caching:
<https://docs.nginx.com/nginx-ingress-controller/configuration/global-configuration/configmap-resource/>
<https://www.nginx.com/blog/nginx-caching-guide/>
- Blue/Green A/B Testing:
<https://github.com/nginxinc/kubernetes-ingress/tree/master/examples-of-custom-resources/traffic-splitting>
<https://www.nginx.com/blog/performing-a-b-testing-nginx-plus/>
<https://www.nginx.com/blog/dynamic-a-b-testing-with-nginx-plus/>

Workshop Wrap-Up

You have completed all the lab exercises in the workshop. Do a final visual check on your Plus Dashboard, and check your Grafana dashboards, what do you see? These tools should show where you finished with statistics and graphs that match your last few lab exercises.

During the Workshop, you learned the following Nginx, Ingress, and Kubernetes topics and completed the following lab exercises:

1. Verify Nginx Ingress Controller is up and running.
 2. Configure access to the NginxPlus Dashboard for monitoring real-time statistics.
 3. Deploy the Café demo application for coffee/tea services.
 4. Add the Bar application and Virtual Server.
 5. Run a load test on your Ingress Controller and the Cafe application.
 6. Scale your Cafe application, and Nginx Ingress up and down, under load without errors.
 7. Change Nginx logging to help troubleshoot pod performance issues.
 8. Set up and run Prometheus and Grafana with Helm, to monitor your cluster, apps and Ingress Controller.
 9. Launch a new application, JuiceShop, and test it.
 10. Enable some of the Advanced features of Nginx Plus, like Active Healthchecks, Caching, Sorry pages, mTLS, and Blue-Green split client testing.
-

Thank You for your time and attention. Please provide feedback to your Workshop hosts when asked, it is important for us to hear your feedback so we can continually improve this workshop.

This completes this Lab.

| Click on Preview below for a Overview on additional workshops from NGINX !!

Authors

- Chris Akker - Technical Solutions Architect @ F5, Inc.
 - Shouvik Dutta - Technical Solutions Architect @ F5, Inc.
-

Navigate to ([Preview](#) | [Main Menu](#))