

Project Details Are From Page 09

Ingress

Ingress Networking

Ingress networking in Kubernetes refers to the way external access to services within a Kubernetes cluster is managed. It provides a powerful mechanism to define rules that allow external HTTP and HTTPS traffic to reach your services. An Ingress resource defines these rules and allows for more complex routing, such as load balancing, SSL termination, and name-based virtual hosting.

Key Concepts

1. **Ingress Resource:** A set of rules that allow inbound connections to reach the cluster services.
2. **Ingress Controller:** A component that implements the Ingress resource, typically deployed as a pod within the cluster. It interprets the Ingress rules and routes traffic accordingly.
3. **Ingress Rules:** Define the routing of HTTP/HTTPS traffic to various services within the cluster based on host, path, etc.
4. **Backend Services:** Kubernetes services that Ingress resources route traffic to.

1. Start Minikube with Ingress Addon

Ensure that Minikube is started with the Ingress addon enabled.

```
minikube start --addons=ingress
```

2. Verify Ingress Controller

Check that the NGINX Ingress controller pods are running.

```
kubectl get pods -n kube-system | grep nginx
```

You should see the NGINX Ingress controller pods listed.

3. Create Sample Services and Deployments

Create sample deployments and services for demonstration.

```
# frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: nginx
          ports:
            - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```
# backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: hashicorp/http-echo
```

```

    args:
      - "-text=Hello from backend"
    ports:
      - containerPort: 5678
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5678

```

Apply these YAML files to create the deployments and services.

```

kubectl apply -f frontend-deployment.yaml
kubectl apply -f backend-deployment.yaml

```

4. Create an Ingress Resource

Define an Ingress resource to route traffic to these services.

```

# ingress-resource.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
    - host: myapp.local
      http:
        paths:
          - path: /frontend
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 80
          - path: /backend
            pathType: Prefix

```

```
backend:
  service:
    name: backend-service
  port:
    number: 80
```

Apply the Ingress resource.

```
kubectl apply -f ingress-resource.yaml
```

5. Update `/etc/hosts`

Add the hostname defined in the Ingress resource to your `/etc/hosts` file pointing to the Minikube IP.

```
sudo nano /etc/hosts
```

Add the following line (replace `<minikube-ip>` with the actual Minikube IP):

```
<minikube-ip> myapp.local
```

Get the Minikube IP using:

```
minikube ip
```

6. Access the Services

You should now be able to access the services via your browser or `curl`.

```
curl http://myapp.local/frontend
curl http://myapp.local/backend
```

Advanced Use Cases

1. Load Balancing with Sticky Sessions

Scenario: Distribute traffic across multiple instances of a service while maintaining session affinity.

Example:

Ingress Resource with Annotations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sticky-ingress
  annotations:
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/session-cookie-name: "route"
spec:
  rules:
  - host: sticky.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: sticky-service
            port:
              number: 80
```

2. Path Rewriting

Scenario: Route traffic to a different path in the backend service.

Example:

Ingress Resource with Path Rewriting:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: rewrite-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: rewrite.local
    http:
      paths:
      - path: /oldpath/(.*)
        pathType: Prefix
        backend:
          service:
            name: new-service
            port:
```

number: 80

nginx.ingress.kubernetes.io/rewrite-target: /

- **Description:** Replaces the path with the specified rewrite target (/). It effectively removes the matched part of the URL path.
- **Example:** If the original request path is /oldpath/something, it will be rewritten to /something before forwarding to the backend service.

nginx.ingress.kubernetes.io/rewrite-target: /newpath

- **Description:** Replaces the path with the specified rewrite target (/newpath). It substitutes the matched part of the URL path with the specified new path.
- **Example:** If the original request path is /oldpath/something, it will be rewritten to /newpath/something before forwarding to the backend service.

nginx.ingress.kubernetes.io/rewrite-target: /\$1

- **Description:** Uses a capture group (\$1) from the original path's regular expression match to dynamically construct the rewritten path.
- **Example:** If your Ingress path definition includes regex capturing groups like path: /oldpath/(.*), and rewrite-target: /\$1 is specified, requests to /oldpath/something will be rewritten to /something.

Key Features of TLS:

1. **Encryption:** TLS encrypts data to ensure that it remains private and secure during transmission. This prevents unauthorized parties from intercepting and reading the data.
2. **Authentication:** TLS supports server-side and optional client-side authentication using digital certificates. This ensures that the parties involved in the communication are who they claim to be.
3. **Compatibility:** TLS is widely supported and used across various applications and services, including web browsers, email clients, instant messaging, and more.

TLS Handshake Process:

TLS communication begins with a handshake process, where the client and server negotiate parameters for secure communication:

- **ClientHello:** The client sends a message containing the TLS version, supported cipher suites, and a random number.
- **ServerHello:** The server responds with its chosen TLS version, cipher suite, and a random number.
- **Certificate Exchange:** The server sends its digital certificate to prove its identity (if required).

- **Key Exchange:** The client and server agree on a shared secret key to be used for symmetric encryption during the session.
- **Finished:** Both parties exchange finished messages to confirm that the handshake was successful and communication can proceed securely.

Usage:

TLS is commonly used to secure HTTP connections (HTTPS), ensuring that sensitive information such as login credentials, payment details, and personal data transmitted over the internet remains confidential and integral.

Create a Secret for TLS Certificate

```
kubectl create secret tls tls-secret --cert=path/to/tls.crt --key=path/to/tls.key
```

Ingress Resource:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - myapp.local
    secretName: my-tls-secret
  rules:
  - host: myapp.local
    http:
      paths:
      - path: /frontend
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
      - path: /backend
```

```
pathType: Prefix
backend:
  service:
    name: backend-service
    port:

      number: 80
```

Project Overview

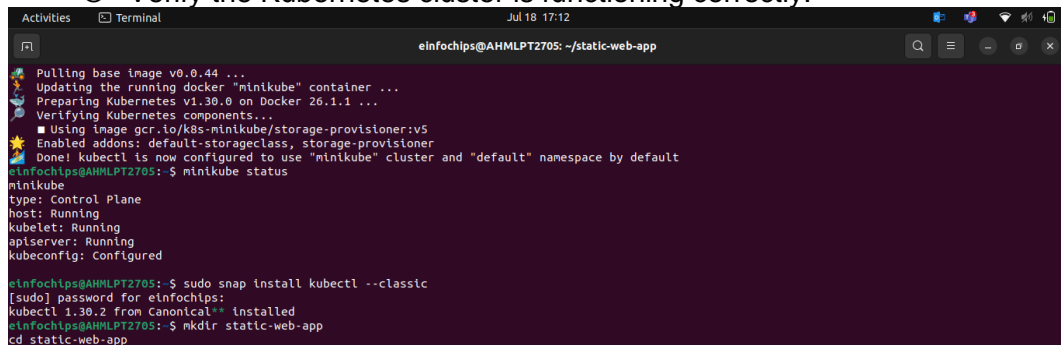
Participants are required to deploy a simple static web application on a Kubernetes cluster using Minikube, set up advanced ingress networking with URL rewriting and sticky sessions, and configure horizontal pod autoscaling to manage traffic efficiently. The project will be divided into stages, with each stage focusing on specific aspects of Kubernetes ingress, URL rewriting, sticky sessions, and autoscaling.

Requirements and Deliverables

Stage 1: Setting Up the Kubernetes Cluster and Static Web App

1. Set Up Minikube:

- Ensure Minikube is installed and running on the local Ubuntu machine.
- Verify the Kubernetes cluster is functioning correctly.



```
einfochips@AHMLPT2705: ~/static-web-app
Pulling base image v0.0.44 ...
Updating the running docker "minikube" container ...
Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
  Enabled addons: default-storageclass, storage-provisioner
Done! kubectrl is now configured to use "minikube" cluster and "default" namespace by default
einfochips@AHMLPT2705: $ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
einfochips@AHMLPT2705: $ sudo snap install kubectrl --classic
[sudo] password for einfochips:
kubectrl 1.30.2 from Canonical✓ installed
einfochips@AHMLPT2705: $ mkdir static-web-app
cd static-web-app
```

- Start Minikube (minikube start)
 - Create a directory named static-web-api in the current working directory (mkdir static-web-api)
- #### 2. Deploy Static Web App:
- Create a Dockerfile for a simple static web application (e.g., an HTML page served by Nginx).
 - Build a Docker image for the static web application.
 - Push the Docker image to Docker Hub or a local registry.


```
Activities Terminal Jul 18 17:17
einfochips@AHMLPT2705: ~/static-web-app

cd static-web-app
einfochips@AHMLPT2705: ~/static-web-app$ nano index.html
einfochips@AHMLPT2705: ~/static-web-app$ nano Dockerfile
einfochips@AHMLPT2705: ~/static-web-app$ docker build -t static-web-app .
[+] Building 0.1s (7/7) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 28
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 359B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [internal] load build context
=> => transferring context: 277B
=> [1/2] FROM docker.io/library/nginx:latest
=> [2/2] COPY index.html /usr/share/nginx/html/index.html
=> exporting to image
=> => exporting layers
=> => writing image sha256:e0184ac496c3575b914dfb6acedbf697e59e96c959a4b506269d821a901937a5
=> => naming to docker.io/library/static-web-app
einfochips@AHMLPT2705: ~/static-web-app$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/einfochips/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
einfochips@AHMLPT2705: ~/static-web-app$ docker tag static-web-app neelpatel5270/static-web-app:latest
einfochips@AHMLPT2705: ~/static-web-app$ docker push neelpatel5270/static-web-app:latest
The push refers to repository [docker.io/neelpatel5270/static-web-app]
255626046736: Pushed
56b6d3be75f9: Mounted from library/nginx
0c6c257920c8: Mounted from library/nginx
92d0d4e97019: Mounted from library/nginx
7190c87a0e8a: Mounted from library/nginx
933a3ce2c78a: Mounted from library/nginx
32cfa91376f: Mounted from library/nginx
32148f9f6c5a: Mounted from library/python
latest: digest: sha256:0a1b5e372fb92e9f5563a4a5a355dade6a20cefef433e46c3392100c2d2afb43 size: 1985
einfochips@AHMLPT2705: ~/static-web-app$
```

Create an `index.html` file in the same directory

Build the Docker Image

docker login

Then push the image

3. Kubernetes Deployment:

- Write a Kubernetes deployment manifest to deploy the static web application.
- Write a Kubernetes service manifest to expose the static web application within the cluster.
- Apply the deployment and service manifests to the Kubernetes cluster.

```
Activities Terminal Jul 18 17:21
einfochips@AHMLPT2705: ~/static-web-app

=> => transferring context: 28
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 359B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [internal] load build context
=> => transferring context: 277B
=> [1/2] FROM docker.io/library/nginx:latest
=> [2/2] COPY index.html /usr/share/nginx/html/index.html
=> exporting to image
=> => exporting layers
=> => writing image sha256:e0184ac496c3575b914dfb6acedbf697e59e96c959a4b506269d821a901937a5
=> => naming to docker.io/library/static-web-app
einfochips@AHMLPT2705: ~/static-web-app$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/einfochips/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
einfochips@AHMLPT2705: ~/static-web-app$ docker tag static-web-app neelpatel5270/static-web-app:latest
einfochips@AHMLPT2705: ~/static-web-app$ docker push neelpatel5270/static-web-app:latest
The push refers to repository [docker.io/neelpatel5270/static-web-app]
255626046736: Pushed
56b6d3be75f9: Mounted from library/nginx
0c6c257920c8: Mounted from library/nginx
92d0d4e97019: Mounted from library/nginx
7190c87a0e8a: Mounted from library/nginx
933a3ce2c78a: Mounted from library/nginx
32cfa91376f: Mounted from library/nginx
32148f9f6c5a: Mounted from library/python
latest: digest: sha256:0a1b5e372fb92e9f5563a4a5a355dade6a20cefef433e46c3392100c2d2afb43 size: 1985
einfochips@AHMLPT2705: ~/static-web-app$ nano deployment.yaml
einfochips@AHMLPT2705: ~/static-web-app$ nano service.yaml
einfochips@AHMLPT2705: ~/static-web-app$ kubectl apply -f deployment.yaml
deployment.apps/static-web-app-deployment created
einfochips@AHMLPT2705: ~/static-web-app$ kubectl apply -f service.yaml
service/static-web-app-service created
einfochips@AHMLPT2705: ~/static-web-app$
```

Deliverables:

- Dockerfile for the static web app
- Docker image URL
- Kubernetes deployment and service YAML files

- Create a file `deployment.yaml`
- Create a file `service.yaml`
- Apply the Deployment and Service Manifests
- Minikube IP address

Stage 2: Configuring Ingress Networking

4. Install and Configure Ingress Controller:

- Install an ingress controller (e.g., Nginx Ingress Controller) in the Minikube cluster.
- Verify the ingress controller is running and accessible.

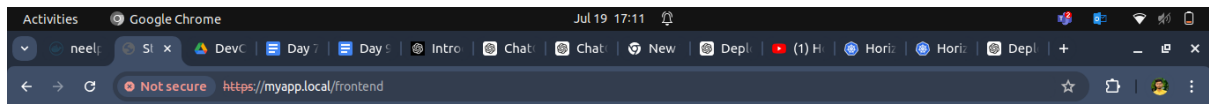
5. Create Ingress Resource:

- Write an ingress resource manifest to route external traffic to the static web application.
- Configure advanced ingress rules for path-based routing and host-based routing (use at least two different hostnames and paths).
- Implement TLS termination for secure connections.
- Configure URL rewriting in the ingress resource to modify incoming URLs before they reach the backend services.
- Enable sticky sessions to ensure that requests from the same client are directed to the same backend pod.

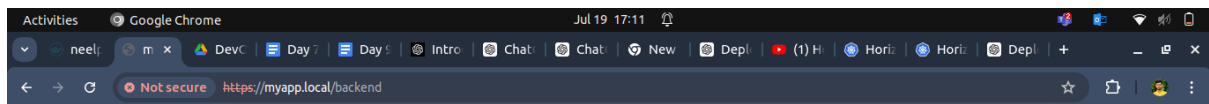
```

einfochips@AHMLPT2705: ~/static-web-app
Labels: <none>
Annotations: <none>
CreationTimestamp: Thu, 18 Jul 2024 23:02:18 +0530
Reference: Deployment/static-web-page
Metrics: ( current / target )
  resource cpu on pods (as a percentage of request): <unknown> / 50%
Min replicas: 2
Max replicas: 5
Deployment pods: 2 current / 0 desired
Conditions:
  Type           Status  Reason
  ----           -
  AbleToScale    True    SucceededGetScale
  ScalingActive  False   FailedGetResourceMetric
Message: the HPA was unable to compute the replica count: failed to get cpu utilization: unable to get metric
s for resource cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (get pods.metrics.k8s.io)
Events:
  Type      Reason      Age      From      Message
  ----      -
  Warning   FailedGetScale  6m10s (x221 over 61m) horizontal-pod-autoscaler deployments/scale.apps "static-web-page" not found
  Warning   FailedGetResourceMetric  70s (x6 over 2m25s) horizontal-pod-autoscaler failed to get cpu utilization: unable to get metrics for resourc
e cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (get pods.metrics.k8s.io)
einfochips@AHMLPT2705:~/static-web-app$ kubectl delete pod load-generator
pod "load-generator" deleted
einfochips@AHMLPT2705:~/static-web-app$ curl http://myapp.local/frontend
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Static Web App</title>
</head>
<body>
  <h1>Hello from Nginx!</h1>
</body>
</html>
Hello from backend
einfochips@AHMLPT2705:~/static-web-app$

```



Hello from Nginx!



Hello from backend

Deliverables:

- Ingress controller installation commands/scripts
- Ingress resource YAML file with advanced routing, TLS configuration, URL rewriting, and sticky sessions
- minikube addons enable ingress

- Create a Kubernetes Secret to store the TLS certificate
- `kubectl create secret tls tls-secret --cert=tls.crt --key=tls.key`
- `nano ingress-rewriting.yaml`
- `kubectl apply -f ingressrewring.yaml`
- Create a `ingress.yaml`
- Apply the Ingress Manifest
- Create a `deployment.yaml`
- Create a `service.yaml`

Stage 3: Implementing Horizontal Pod Autoscaling

6. Configure Horizontal Pod Autoscaler:

- Write a horizontal pod autoscaler (HPA) manifest to automatically scale the static web application pods based on CPU utilization.
- Set thresholds for minimum and maximum pod replicas.

```
</body>
</html>
Hello from backend
einfochips@AHMLPT2705:~/static-web-app$ kubectl autoscale deployment frontend --cpu-percent=50 --min=2 --max=4
horizontalpodautoscaler.autoscaling/frontend autoscaled
einfochips@AHMLPT2705:~/static-web-app$ kubectl autoscale deployment backend --cpu-percent=50 --min=2 --max=4
horizontalpodautoscaler.autoscaling/backend autoscaled
einfochips@AHMLPT2705:~/static-web-app$
```

○

7. Stress Testing:

- Perform stress testing to simulate traffic and validate the HPA configuration.
- Monitor the scaling behavior and ensure the application scales up and down based on the load.

```
1862 rm hpa.yaml.save
1863 rm load-generator.yaml
1864 kubectl delete service static-web-app-service
1865 ls
1866 nano deployment.yaml
1867 nano service.yaml
1868 nano deployment.yaml
1869 nano hpa.yaml
1870 kubectl apply -f deployment.yaml
1871 kubectl apply -f hpa.yaml
1872 nano hpa.yaml
1873 kubectl apply -f hpa.yaml
1874 nano hpa.yaml
1875 kubectl apply -f service.yaml
1876 minikube service my-app-service --url
1877 kubectl run -i --tty --rm load-generator --image=busybox --restart=Never -- /bin/sh
1878 kubectl run -i --tty --rm load-generator-new --image=busybox --restart=Never -- /bin/sh
1879 history
einfochips@AHMLPT2705:~/static-web-app$
```

- Horizontal Pod Autoscaler (HPA) for your static web application based on CPU utilization, you'll need to create an HPA manifest. This manifest will define the rules for scaling your application pods automatically.
- apply the HPA manifest
- `kubectl get hpa`
- This command will show you the current status of the HPA, including the current number of replicas and the target metrics.
- `kubectl get pods -w`

This command will continuously display the status of the pods in your deployment, allowing you to see how the number of pods changes in response to the load.

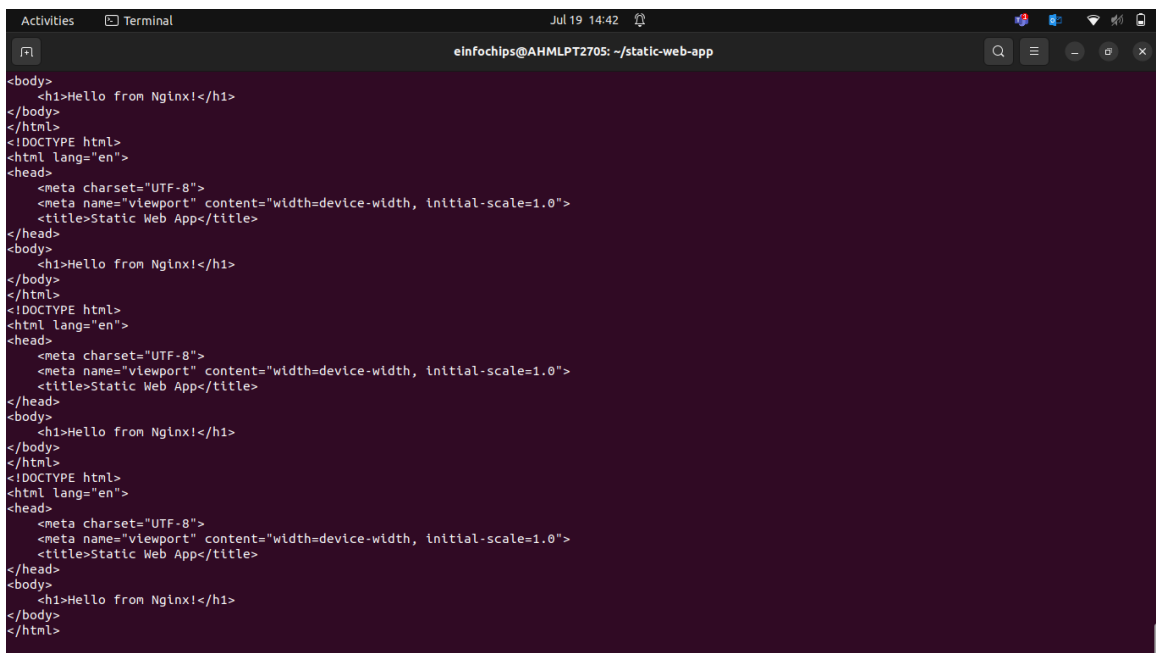
- **Scaling Up:** As the load increases, the HPA should trigger the creation of additional pods to handle the increased traffic.
- **Scaling Down:** Once the load decreases, the HPA should reduce the number of pods back to the minimum defined in the manifest.
- `kubectl get hpa`

This output indicates that the HPA is targeting 50% CPU utilization, and currently, the CPU utilization is at 70%, so it has scaled up to 5 replicas.

This output shows the scaling behavior of the pods, with new pods being created and running as the load increases.

Deliverables:

- Horizontal pod autoscaler YAML file
- Documentation or screenshots of the stress testing process and scaling behavior



```
Activities Terminal Jul 19 14:42 einfochips@AHMLPT2705: ~/static-web-app
<body>
  <h1>Hello from Nginx!</h1>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Static Web App</title>
</head>
<body>
  <h1>Hello from Nginx!</h1>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Static Web App</title>
</head>
<body>
  <h1>Hello from Nginx!</h1>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Static Web App</title>
</head>
<body>
  <h1>Hello from Nginx!</h1>
</body>
</html>
```

```
Activities Terminal Jul 19 16:25 einfochips@AHMLPT2705: ~/static-web-app

Name: system:service-account-issuer-discovery
Subjects:
  Kind Name Namespace
  ----
  Group system:serviceaccounts

Name: system:volume-scheduler
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: system:volume-scheduler
Subjects:
  Kind Name Namespace
  ----
  User system:kube-scheduler

einfochips@AHMLPT2705:~/static-web-app$ kubectl top pods
error: Metrics API not available
error: Metrics API not available
einfochips@AHMLPT2705:~/static-web-app$ ls
backend-deployment.yaml Dockerfile hpa.yaml ingress-resource.yaml service.yaml sticky-ingress.yaml tls.key
deployment.yaml frontend-deployment.yaml index.html ingress-rewriting.yaml static-web-page-deployment.yaml tls.crt
einfochips@AHMLPT2705:~/static-web-app$ nano ingress-resource.yaml
einfochips@AHMLPT2705:~/static-web-app$ nano deployment.yaml
einfochips@AHMLPT2705:~/static-web-app$ kubectl top nodes
error: Metrics API not available
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
backend Deployment/backend cpu: <unknown>/50% 2 4 2 3h10m
frontend Deployment/frontend cpu: <unknown>/50% 2 4 2 3h11m
my-app-hpa Deployment/my-app cpu: <unknown>/2% 2 5 2 54m
static-web-app-hpa Deployment/static-web-app cpu: <unknown>/50% 1 10 2 16h
einfochips@AHMLPT2705:~/static-web-app$ nano deployment.yaml
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hbja
error: the server doesn't have a resource type "hbja"
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
```

Stage 4: Final Validation and Cleanup

8. Final Validation:

- Validate the ingress networking, URL rewriting, and sticky sessions configurations by accessing the web application through different hostnames and paths.
- Verify the application's availability and performance during different load conditions.

9. Cleanup:

- Provide commands or scripts to clean up the Kubernetes resources created during the project (deployments, services, ingress, HPA).

Deliverables:

- Final validation report documenting the testing process and results
- Cleanup commands/scripts

```

Activities  Terminal  Jul 19 17:00  einfochips@AHMLPT2705: ~/static-web-app

kubernetes      ClusterIP  10.96.0.1      <none>      443/TCP      23h
my-app-service  LoadBalancer  10.105.125.32 <pending>  80:32228/TCP 3h4m
einfochips@AHMLPT2705:~/static-web-app$ kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
backend   2/2     2            2           6h16m
frontend  2/2     2            2           6h16m
my-app     2/2     2            2           3h7m
einfochips@AHMLPT2705:~/static-web-app$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/my-app-hpa created
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
NAME      REFERENCE      TARGETS      MINPODS  MAXPODS  REPLICAS  AGE
my-app-hpa  Deployment/my-app  cpu: <unknown>/50%  2        5        0         7s
einfochips@AHMLPT2705:~/static-web-app$ nano deployment.yaml
einfochips@AHMLPT2705:~/static-web-app$ nano hpa.yaml
einfochips@AHMLPT2705:~/static-web-app$ kubectl apply -f deployment.yaml
deployment.apps/my-app configured
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
NAME      REFERENCE      TARGETS      MINPODS  MAXPODS  REPLICAS  AGE
my-app-hpa  Deployment/my-app  cpu: 0%/50%  2        5        3         2m38s
einfochips@AHMLPT2705:~/static-web-app$ nano hpa.yaml
einfochips@AHMLPT2705:~/static-web-app$ kubectl get service
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP  PORT(S)      AGE
backend-service  ClusterIP  10.101.109.109  <none>      80/TCP       17h
frontend-service  ClusterIP  10.96.30.89     <none>      80/TCP       17h
kubernetes      ClusterIP  10.96.0.1       <none>      443/TCP      23h
my-app-service  LoadBalancer  10.105.125.32  <pending>  80:32228/TCP 3h10m
einfochips@AHMLPT2705:~/static-web-app$ nano hpa.yaml
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
NAME      REFERENCE      TARGETS      MINPODS  MAXPODS  REPLICAS  AGE
my-app-hpa  Deployment/my-app  cpu: 0%/50%  2        5        2         5m58s
einfochips@AHMLPT2705:~/static-web-app$ kubectl get hpa
NAME      REFERENCE      TARGETS      MINPODS  MAXPODS  REPLICAS  AGE
my-app-hpa  Deployment/my-app  cpu: 105%/50%  2        5        5         6m49s
einfochips@AHMLPT2705:~/static-web-app$ nano hpa.yaml
einfochips@AHMLPT2705:~/static-web-app$ nano deployment.yaml
einfochips@AHMLPT2705:~/static-web-app$ nano service.yaml
einfochips@AHMLPT2705:~/static-web-app$ nano service.yaml
einfochips@AHMLPT2705:~/static-web-app$

```