Tigera

Customer Success Technical Assessment

Next Interview Structure

- 30min Technical Challenge demo
- 15min presentation
- Q&A throughout the interview

Technical Challenge Setup:

Install a kubernetes cluster using the distribution and the environment of your choice (your laptop or cloud) including 1x master and 2x nodes. Do not use managed kubernetes such as EKS, AKS or GKE. **Cluster with 1 Master and 2 Worker nodes have been deployed using Kubeadm with Calico as CNI of choice.**

- Install calico, bookinfo app (attached), and an alpine pod for testing. **Calico deployed and a pod running alpine image is created.**
- The bookinfo app is managed by a team called dev1 : **App deployed in a namespace called dev1**.
- Expose bookinfo product page using the method of your choice to users accessing it from outside of the cluster from a well-defined network range.
- Implement granular calico policies restricting communication among bookinfo micro-services to the bare minimum. Define your strategy with respect to implementing ingress only, egress only, or ingress and egress controls.

Technical Challenge Demo Process:

- **Test access to bookinfo micro-services from within dev1 environment**:
  Bookinginfo app deployed and then access tested from within the dev1 namespace as shown below:

```
[ec2-user@ip-10-0-4-58 ~]$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/alpine                          1/1     Running   0          21m
pod/details-v1-c684c7555-bv78p      1/1     Running   0          4h14m
pod/dnsutils                        1/1     Running   0          50m
pod/productpage-v1-bcc6748f7-dh26s  1/1     Running   0          4h14m
pod/ratings-v1-64d59779-zrb6p       1/1     Running   0          4h14m
pod/reviews-v1-6cc9f745f7-7c779     1/1     Running   0          4h14m
pod/reviews-v2-69d467bc99-tkq2h     1/1     Running   0          4h14m
pod/reviews-v3-7bf7d59b7c-4wczw     1/1     Running   0          4h14m

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/details       ClusterIP   10.100.147.6     <none>        9080/TCP   4h14m
service/productpage   ClusterIP   10.100.213.134   <none>        9080/TCP   4h14m
service/ratings       ClusterIP   10.102.120.57    <none>        9080/TCP   4h14m
service/reviews       ClusterIP   10.110.100.92    <none>        9080/TCP   4h14m
```

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/details-v1              1/1     1            1           4h14m
deployment.apps/productpage-v1          1/1     1            1           4h14m
deployment.apps/ratings-v1              1/1     1            1           4h14m
deployment.apps/reviews-v1              1/1     1            1           4h14m
deployment.apps/reviews-v2              1/1     1            1           4h14m
deployment.apps/reviews-v3              1/1     1            1           4h14m

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/details-v1-c684c7555          1         1         1       4h14m
replicaset.apps/productpage-v1-bcc6748f7      1         1         1       4h14m
replicaset.apps/ratings-v1-64d59779           1         1         1       4h14m
replicaset.apps/reviews-v1-6cc9f745f7         1         1         1       4h14m
replicaset.apps/reviews-v2-69d467bc99         1         1         1       4h14m
replicaset.apps/reviews-v3-7bf7d59b7c         1         1         1       4h14m
```

Now to test the connectivity between microservices in the namespace dev1 we will do a curl from one pod to another as shown below:
We will get exec into the ratings pod and curl the product pod from there to check the access
kubectl exec -it ratings-v1-64d59779-zrb6p -- /bin/sh

```
[ec2-user@ip-10-0-4-58 ~]$ kubectl exec -it ratings-v1-64d59779-zrb6p -- /bin/sh
# curl -sS productpage:9080/productpage
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Bookstore App</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap-theme.min.css">
```

This confirms the connectivity test from one microservice to another in the namespace dev1.

To test the connectivity further we will create an ALPINE pod in the same namespace and test the same from there.

ec2-user@ip-10-0-4-58:~

```
[ec2-user@ip-10-0-4-58 ~]$ kubectl run alpine --image=alpine --command -- sleep infinity
pod/alpine created
[ec2-user@ip-10-0-4-58 ~]$ kubectl exec -it alpine -- /bin/sh
/ #
```

```
/ # curl -sS productpage:9080/productpage
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Bookstore App</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap-theme.min.css">

  </head>
  <body>
```

This completes the task mentioned above.

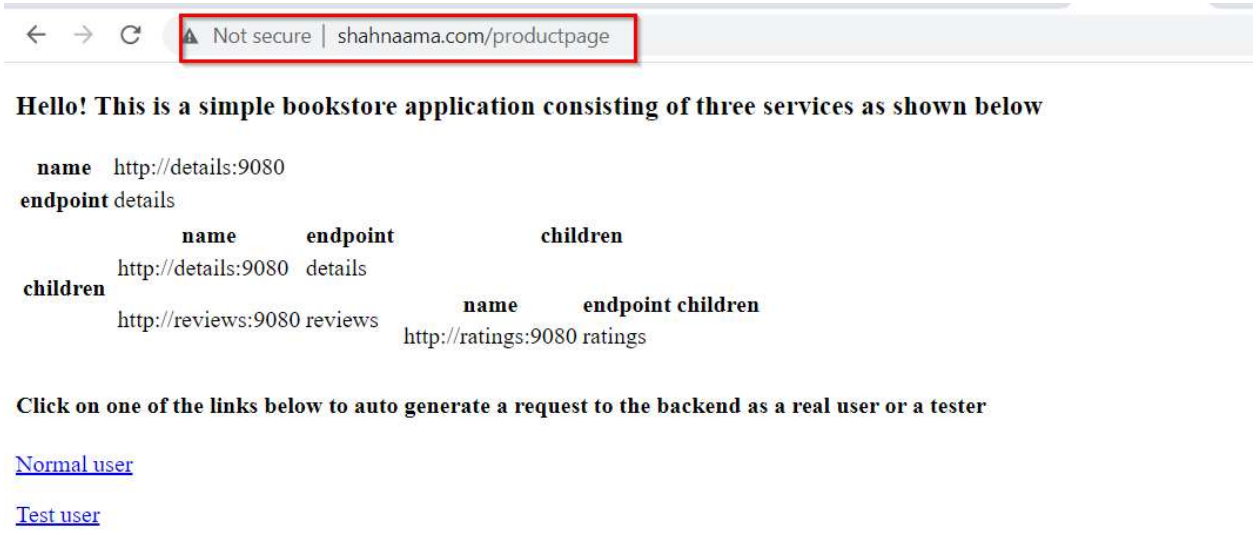
- **Test access to bookinfo micro-services from within the cluster, outside of dev1 environment.**
  we will create another Alpine Pod in the default namespace and will try to access the same product landing page as before from there using the service name.

```
/ # curl -sS productpage.dev1.svc.cluster.local:9080/productpage
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Bookstore App</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="static/bootstrap/css/bootstrap-theme.min.css">

  </head>
  <body>
```

We will restrict this access later-on using network policies.

- Test access to bookinfo micro-services from outside of the cluster: Deployed an nginx ingress controller and created an ingress resource to allow path based routing exposing different services.
  Tested it as below:
  curl shahnaama.com/productpage | grep title

```
[ec2-user@ip-10-0-4-58 ~]$ curl shahnaama.com/productpage | grep title
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1683  100  1683    0     0   205k      0 --:--:-- --:--:-- --:--:--  205k
    <title>Simple Bookstore App</title>
[ec2-user@ip-10-0-4-58 ~]$
```



- Explain the details of the configuration and testing: As explained in above scenarios.
- Explain the details of the underlying routing that allows pods communication

CALICO CREATES AND MANAGES A LAYER 3 NETWORK THAT PROVIDES INTER-POD COMMUNICATION IN THE KUBERNETES CLUSTER. IT PROVIDES ROUTABLE IP ADDRESSES TO PODS THAT ENABLE EASIER INTEROPERABILITY. CALICO USES THE FOLLOWING COMPONENTS TO ACHIEVE THIS.

CALICO/NODE: THE AGENT THAT RUNS AS PART OF THE CALICO DAEMONSET POD.
IT MANAGES INTERFACES, ROUTES, AND STATUS REPORTING OF THE NODE AND ENFORCES POLICIES.

- ❑ BIRD: A BGP CLIENT THAT BROADCASTS ROUTES THAT ARE PROGRAMMED BY FELIX
- ❑ ETCD: AN OPTIONAL DISTRIBUTED DATASTORE
- ❑ CALICO CONTROLLER: THE CALICO POLICY CONTROLLER:
- ❑ WHEN ONE OF THE NGINX PODS IS SCHEDULED ON THE KUBERNETES NODE, FELIX WILL CREATE A VIRTUAL INTERFACE WITH THE CALI PREFIX AND ASSIGNS IT A /32 IP ADDRESS.

- ❑ THE BIRD BGP DAEMON REALIZES THAT THERE IS A NEW NETWORK INTERFACE THAT HAS COME UP AND IT ADVERTISES THAT TO THE OTHER PEERS.
- ❑ THE CALICO IP ADDRESS MANAGEMENT (IPAM) IS RESPONSIBLE FOR IP ADDRESS MANAGEMENT FOR PODS ON EACH NODE OF THE KUBERNETES CLUSTER.
- ❑ MANAGES IPPOOLS FOR ASSIGNING IP ADDRESSES.
- ❑ CALICO DOES DYNAMIC ADDRESS MANAGEMENT BY ALLOCATING A /26 ADDRESS BLOCK TO EACH NODE WHEN THE CALICO NODE RUNS SUCCESSFULLY.

- Explain the challenges you faced and how you resolved them.

  Roadblocks while performing the lab:

  1. The cluster was set-up manually using virtual machines on AWS. While, creating the ingress controller it was not creating loadbalancer service with a public IP to expose the nginx-ingress controller – not supported in manual Kubernetes Cluster but only when using EKS (so the external IP of the loadbalancer service was in pending status). Solved the problem by manually deploying a bare metal load balancer using MetalLB and used it to assign an IP to the ingress controller service.

  2. Calico deployment was failing with timeouts during installation. Tweaked the security group in AWS to allow port 179 across nodes and restarted the process which solved the problem.

  3. The nginx ingress controller was not exposing the app outside the cluster. After troubleshooting for sometime, got to know that we need to change the hostNetwork: true in the deployment spec. Updated the running deployment with the following entry and tested the application again and it started working fine.

Presentation:

Prepare a few slides explaining the following:

- Kubernetes components that interact for deploying an application
- Calico and kubernetes components that interact for provisioning Pod networking
- Calico and kubernetes components that interact for securing access to an application
- Calico, kubernetes and infrastructure components that interact for exposing an application

A powerpoint Slide has been attached.

Kubernetes-Internals.pptx