Troubleshooting Network Failures

Hands On – Troubleshoot a failing Network

I. Deploy the service and pod then identify the svc DNS.

Let's start by deploying our nginx pods and service.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-service.yaml
---
kind: Service
apiVersion: v1
metadata:
  name: doms-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-service.yaml
service/doms-service created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
doms-service      ClusterIP   10.102.116.135   <none>        80/TCP    4s
```

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-pod.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: doms-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-pod.yaml
deployment.apps/doms-deployment created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
doms-deployment-5d59d67564-7t9tp   1/1     Running   0          6s
doms-deployment-5d59d67564-vgbn5   1/1     Running   0          6s
doms-deployment-5d59d67564-zdf8c   1/1     Running   0          6s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

To get the DNS name of our service we need to follow the format according to the Kubernetes documentation:

my-svc.my-namespace.svc.cluster-domain.example

So our DNS address for the service will look something like this:

yourname-service.default.svc.cluster.local

DNS is a service that allows us to resolve hostnames and IP addresses. Let's ensure that our DNS service is working properly with our new pods. If DNS isn't working as needed, we will not be able to utilize certain resources in our cluster to access the external internet.

II. Create a pod that has nslookup to test DNS

Nslookup is a utility used by network engineers to troubleshoot internet connectivity. Essentially, nslookup is like looking up a name and number in a phone book and prints out as such:

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ nslookup 8.8.8.8
8.8.8.8.in-addr.arpa    name = dns.google.

Authoritative answers can be found from:
```

By using nslookup we can ensure our DNS service is working as intended. This manifest creates a pod with all of the DNS utilities for testing.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-dnspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: dnsutils
  namespace: default
spec:
  containers:
  - name: dnsutils
    image: k8s.gcr.io/e2e-test-images/jessie-dnsutils:1.3
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-dnspod.yaml
pod/dnsutils unchanged
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
dnsutils                            1/1     Running   0          18s
doms-deployment-5d59d67564-7t9tp    1/1     Running   0          13m
doms-deployment-5d59d67564-vgbn5    1/1     Running   0          13m
doms-deployment-5d59d67564-zdf8c    1/1     Running   0          13m
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Now that we have the dnsutils pod deployed, let's SSH into it and get the IP for our service using our hostname.

nslookup yourname-service.default.svc.cluster.local

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl exec -it dnsutils sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future versi
-- [COMMAND] instead.
#
#
# nslookup doms-service.default.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   doms-service.default.svc.cluster.local
Address: 10.102.116.135
```

When you run the command you will see an IP printed at the bottom. For my service this IP is 10.102.116.135.

III. Test Connectivity using Curl

We need to test our connection to the service name on port 80 to ensure the pod is working as expected so that if any other pods or services tried to reach our pods via HTTP, they will have no issues.

Here is how you get a curl command line within a Kubernetes network to test and explore your internal REST endpoints.

To get a prompt of a busybox pod running inside the network, execute the following command.

Bonus: Look into the curl pod's YAML to better understand this deployment. You can use kubectl edit pod curl to view.

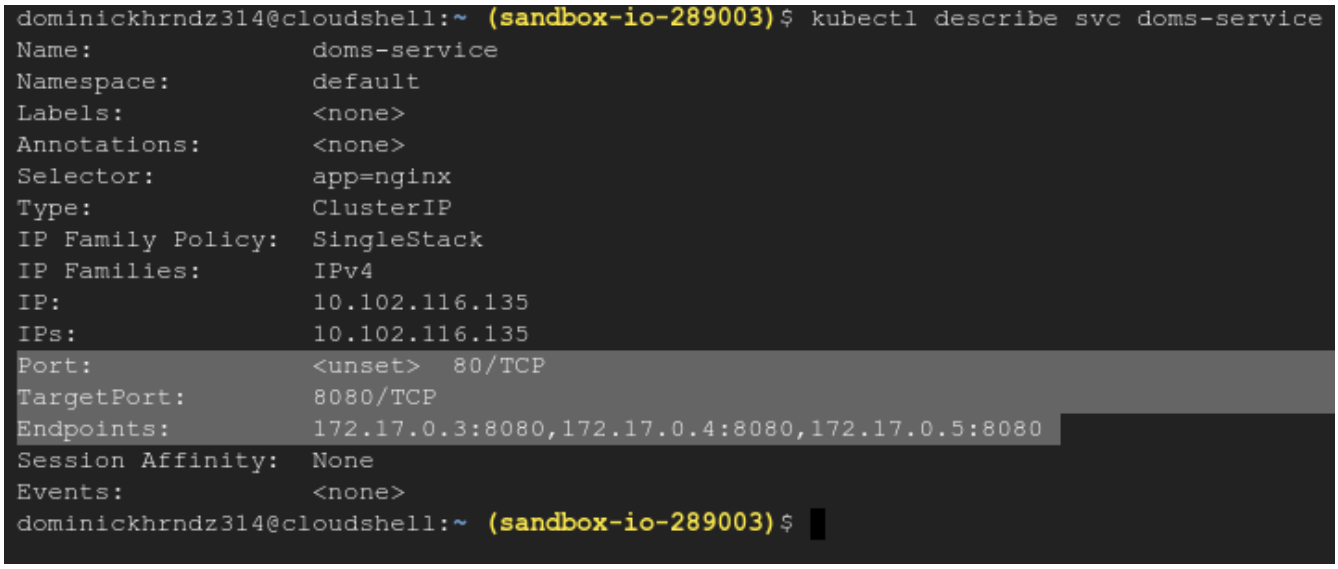
```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl run curl --image=radial/busyboxplus:curl -i --tt
y
If you don't see a command prompt, try pressing enter.
[ root@curl:/ ]$
```

Test connectivity of yourname-service.default.svc.cluster.local on port 80. You should see the curl connection refused.

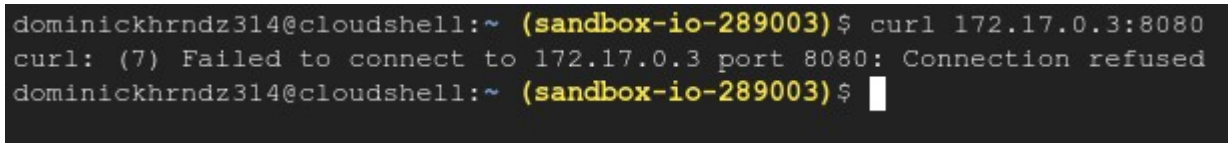
```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl run curl --image=radial/busyboxplus:curl
y
If you don't see a command prompt, try pressing enter.
[ root@curl:/ ]$ curl doms-service.default.svc.cluster.local
curl: (7) Failed to connect to doms-service.default.svc.cluster.local port 80: Connection refused
[ root@curl:/ ]$
```

In your browser, open a second cloud shell.

From here check your service by describing it. You can see the service is listening on port 80 and has an endpoint list with a target of port 8080.



Let's test curl against a pod from the *Endpoint* field.



We can see that our connection is being refused because port 8080 isn't listening. Let's check that pods config by first listing it in the kubectl get pods -o wide command. This will allow us to identify the pod by the IP address we just used.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   REA
DINESS GATES
curl                                1/1     Running   0          8m41s   172.17.0.7   minikube   <none>           <none>
dnsutils                            1/1     Running   0          25m     172.17.0.6   minikube   <none>           <none>
doms-deployment-5d59d67564-7t9tp    1/1     Running   0          38m     172.17.0.4   minikube   <none>           <none>
doms-deployment-5d59d67564-vgbn5    1/1     Running   0          38m     172.17.0.3   minikube   <none>           <none>
doms-deployment-5d59d67564-zdf8c    1/1     Running   0          38m     172.17.0.5   minikube   <none>           <none>
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

To simplify the process we can use a single command to check the listening port on our pod by first describing, then “grep-ing” or grabbing the port.

kubectl describe pod <podname> | grep -i "port:"

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl describe po doms-deployment-5d59d67564-vgbn5 | grep -i "p
ort:"
    Port:         80/TCP
    Host Port:    0/TCP
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Notice that our service is trying to forward traffic to port 8080 as the target port on the container, but the container is only listening on port 80. This is our issue!

IV. Resolve the Failure

We need to reconfigure the service and replace the field *targetPort: 8080* with *targetPort: 80.*

Let’s edit the service.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-service.yaml
---
kind: Service
apiVersion: v1
metadata:
  name: doms-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Now lets delete the existing service, re-apply the YAML, and describe the new service.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl delete svc doms-service
service "doms-service" deleted
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f  doms-service.yaml
service/doms-service created
```

You should now see the targetPort of 80 listed on all endpoints.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl describe svc doms-service
Name:              doms-service
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=nginx
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.103.82.57
IPs:               10.103.82.57
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         172.17.0.3:80,172.17.0.4:80,172.17.0.5:80
Session Affinity:  None
Events:            <none>
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Now go back to the first cloud shell that's running your curl command and retest. If your output looks like the following, you have completed this training.

```
[ root@curl:/ ]$ curl doms-service.default.svc.cluster.local
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[ root@curl:/ ]$
```