Lab 7.2 - Debugging the Linkerd Mesh

Overview

In this lab, you'll simulate a failure in the mesh so you can try some tools to debug issues with Linkerd itself. The number of components that make up a service mesh can make debugging it hard. Having an understanding of the tools you can use to identify the root causes of issues in your application will help you in times of trouble.

For the purposes of this lab, you can use the debug sidecar.

1. If you don’t still have an active session from the previous Linkerd lab, perform these steps to reestablish it:

    a. Connect to the VM that hosts your Kubernetes clusters.

    b. To start the cluster that contains the Linkerd service mesh, issue this command:

    ```
    yourname@ubuntu-vm:~$ docker start $(docker ps -a -f
    name=linkerd-control-plane -q)

    8af4c08524df
    ```

    c. To switch the **kind** context to the Linkerd cluster, use this command:

    ```
    yourname@ubuntu-vm:~$ kubectl config use-context
    kind-linkerd

    Switched to context "kind-linkerd".
    ```


2. You will continue to use the booksapp application from the previous lab in this lab. Issue the command below to make sure all of the **Pod**s in your cluster have a status of **Running** and a ready status with the same numbers on the left and right sides of the slash ("/") before continuing. If any of the lines don't show those statuses, wait a few minutes and reissue the command again.

```
yourname@ubuntu-vm:~$ kubectl get pods --all-namespaces

NAMESPACE              NAME
      READY    STATUS   RESTARTS    AGE
            1/1   Running    0            7h
kube-system            coredns-66bff467f8-9m6v8
      1/1   Running    0            7h
linkerd                linkerd-controller-6df458948d-n7tvb
      2/2   Running    0            7h
booksapp               authors-7fb885df4c-4m2s8
      2/2   Running    0            3m23s
booksapp               books-75b8b95bcf-76zwz
      2/2   Running    0            3m23s
booksapp               traffic-7f4758fcb4-ffpwr
      2/2   Running    0            3m23s
booksapp               webapp-86596d7887-f5ff7
      2/2   Running    0            3m23s
linkerd                linkerd-destination-94bcf958f-mtvnq
      2/2   Running    0            7h
[additional output omitted]
```

3. Before you simulate an issue with the mesh, issue the command below and look at the valid output of **linkerd check --proxy**. The **--proxy** flag tells linkerd to only check the status of the data plane proxies that are actually handling requests in your application.

```
yourname@ubuntu-vm:~$ linkerd check --proxy

kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version
√ is running the minimum kubectl version
```



```
linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ controller pod is running
√ can initialize the client
√ can query the control plane API

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ control plane PodSecurityPolicies exist

linkerd-identity
----------------
√ certificate config is valid
√ trust anchors are using supported crypto algorithm
√ trust anchors are within their validity period
√ trust anchors are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust anchor

linkerd-identity-data-plane
---------------------------
√ data plane proxies certificate match CA

linkerd-api
-----------
√ control plane pods are ready
√ control plane self-check
√ [kubernetes] control plane can talk to Kubernetes
√ [prometheus] control plane can talk to Prometheus
```



```
√ tap api service is running

linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

linkerd-data-plane
------------------
√ data plane namespace exists
√ data plane proxies are ready
√ data plane proxy metrics are present in Prometheus
√ data plane is up-to-date
√ data plane and cli versions match

linkerd-addons
--------------
√ 'linkerd-config-addons' config map exists

linkerd-grafana
---------------
√ grafana add-on service account exists
√ grafana add-on config map exists
√ grafana pod is running

Status check results are √
```

4. Now you will simulate an error in the control plane and then see how to debug it. The Linkerd control plane runs in the **`linkerd`** namespace and has many components that are used to run Linkerd. Take a look at the control plane architecture on the Linkerd website to get a better understanding of what each component is responsible for.

5. Re-install the Emoji demo app:

```
yourname@ubuntu-vm:~$ curl --proto '=https' --tlsv1.2 -sSfL
https://run.linkerd.io/emojivoto.yml \ | kubectl apply -f -
```

6. Delete the Books app ingress:

```
yourname@ubuntu-vm:~$ kubectl delete ingress ingress-booksapp -n
booksapp
```


7. Set up the Nginx Ingress to listen to the Emoji app again as in the previous lab, you pointed it to the Books app.

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
—
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: emojivoto
  name: ingress-emojivoto
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/service-upstream: "true"
    consul.hashicorp.com/connect-inject: “true”
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - pathType: Prefix
          path: "/"
          backend:
            service:
              name: web-svc
              port:
                number: 80
      EOF
```

8. Next, deploy the debug sidecar container.

```
yourname@ubuntu-vm:~$ kubectl -n emojivoto get deploy/voting -o yaml
\
  | linkerd inject --enable-debug-sidecar - \
  | kubectl apply -f -

deployment "voting" injected
deployment.apps/voting configured
```

9. Confirm that the debug container is running.

```
yourname@ubuntu-vm:~$ kubectl get pods -n emojivoto -l app=voting-svc
\
```



```
  -o jsonpath='{.items[*].spec.containers[*].name}'

      linkerd-proxy voting-svc linkerd-debug%
```

10. You can now watch the live tshark output from the logs.

```
      yourname@ubuntu-vm:~$ kubectl -n emojivoto logs deploy/voting
linkerd-debug -f

            2208 145.565533637  10.244.0.34 → 10.244.0.9   TLSv1.3 141
Application Data
       2209 145.565583804   10.244.0.9 → 10.244.0.34  TCP 68 35942 →
4191 [ACK] Seq=2410 Ack=49750 Win=64128 Len=0 TSval=1352785030
TSecr=1709083926
       2210 145.565612554  10.244.0.34 → 10.244.0.9   TLSv1.3 3580
Application Data
       2211 145.565633804   10.244.0.9 → 10.244.0.34  TCP 68 35942 →
4191 [ACK] Seq=2410 Ack=53262 Win=63232 Len=0 TSval=1352785030
TSecr=1709083926
       2212 145.960358304  10.244.0.30 → 10.244.0.34  GRPC 131
HEADERS[235]: POST /emojivoto.v1.VotingService/VoteTaco, DATA[235]
(GRPC) (PROTOBUF)
       2213 145.962068263  10.244.0.34 → 10.244.0.34  GRPC 158
HEADERS[235]: POST /emojivoto.v1.VotingService/VoteTaco, DATA[235]
(GRPC) (PROTOBUF)
       2214 145.962636138  10.244.0.34 → 10.244.0.34  HTTP2 98
WINDOW_UPDATE[0], PING[0]
       2215 145.962643221  10.244.0.34 → 10.244.0.34  TCP 68 50742 →
8080 [ACK] Seq=13375 Ack=7407 Win=65536 Len=0 TSval=3942159986
TSecr=3942159986
       2216 145.962707013  10.244.0.34 → 10.244.0.34  HTTP2 85 PING[0]
       2217 145.963630388  10.244.0.34 → 10.244.0.34  GRPC 104
HEADERS[235]: 200 OK, DATA[235], HEADERS[235] (GRPC) (PROTOBUF)
       2218 145.963636721  10.244.0.34 → 10.244.0.34  TCP 68 50742 →
8080 [ACK] Seq=13392 Ack=7443 Win=65536 Len=0 TSval=3942159987
TSecr=3942159987
       2219 145.963942096  10.244.0.34 → 10.244.0.30  GRPC 128
HEADERS[235]: 200 OK, DATA[235], HEADERS[235] (GRPC) (PROTOBUF)
       2220 145.963979013  10.244.0.30 → 10.244.0.34  TCP 68 35066 →
8080 [ACK] Seq=13441 Ack=8441 Win=64256 Len=0 TSval=1570900962
TSecr=3623533114
```



```
      2221 145.964110179  10.244.0.30 → 10.244.0.34  HTTP2 98
WINDOW_UPDATE[0], PING[0]
      2222 145.964734971  10.244.0.34 → 10.244.0.30  HTTP2 85 PING[0]
      2223 145.964758721  10.244.0.30 → 10.244.0.34  TCP 68 35066 →
8080 [ACK] Seq=13471 Ack=8458 Win=64256 Len=0 TSval=1570900963
TSecr=3623533115
```

11. Another option is to Exec directly into the container itself so you can run a tshark command while inside of the container. You can then view and see any errors that may be occurring inside of your application.

```
    yourname@ubuntu-vm:~$ kubectl -n emojivoto exec -it \
  $(kubectl -n emojivoto get pod -l app=voting-svc \
    -o jsonpath='{.items[0].metadata.name}') \
  -c linkerd-debug -- tshark -i any -f "tcp" -V -Y "http.request"

            Interface name: any
         Encapsulation type: Linux cooked-mode capture (25)
         Arrival Time: Jul 28, 2022 12:39:53.187339506 UTC
         [Time shift for this packet: 0.000000000 seconds]
         Epoch Time: 1659011993.187339506 seconds
         [Time delta from previous captured frame: 0.000132792
seconds]
         [Time delta from previous displayed frame: 0.000000000
seconds]
         [Time since reference or first frame: 9.286449546 seconds]
         Frame Number: 154
         Frame Length: 176 bytes (1408 bits)
         Capture Length: 176 bytes (1408 bits)
         [Frame is marked: False]
         [Frame is ignored: False]
         [Protocols in frame: sll:ethertype:ip:tcp:http]
    Linux cooked capture
         Packet type: Unicast to us (0)
         Link-layer address type: 1
         Link-layer address length: 6
         Source: 46:08:50:cf:cf:17 (46:08:50:cf:cf:17)
         Unused: 0000
         Protocol: IPv4 (0x0800)
```

12. You've reached the end of this lab. Stop the Linkerd cluster by running this command:


```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=linkerd-control-plane -q)

8af4c08524df
```

