Lab 7.1 - Timeouts and Retries with Linkerd

Overview

In this lab, you'll be configuring timeouts and retries for service-to-service networking with Linkerd. Timeouts and retries are important for mitigating cascading failures when a microservice is not responding as expected.

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

2. 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
```

3. 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".
```

4. In the previous labs you used the Emojivoto application to demonstrate basic service mesh functionality. Since Linkerd is currently unable to configure retries on gRPC requests, which is what Emojivoto uses, you will use a different example application for this lab. Run the following command to remove the Emojivoto application from your cluster. Note that it may take this command a few minutes to complete executing and return you to a command line prompt.


```
yourname@ubuntu-vm:~$ kubectl delete namespace emojivoto

namespace "emojivoto" deleted
serviceaccount "emoji" deleted
serviceaccount "voting" deleted
serviceaccount "web" deleted
service "emoji-svc" deleted
service "voting-svc" deleted
service "web-svc" deleted
deployment.apps "emoji" deleted
deployment.apps "vote-bot" deleted
deployment.apps "voting" deleted
deployment.apps "web" deleted
```

5. Make sure Linkerd and Nginx Ingress have started and have all resources ready before continuing. The following command will list the **Pod**s in all namespaces. All should have a status of **Running** and a ready status with the same numbers on the left and right sides of the slash ("/"). 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 -n linkerd

NAMESPACE              NAME
linkerd-destination-7df897cf9b-4w49m      4/4     Running   8
(2m42s ago)   20h
linkerd-identity-5f4dbf785d-vxnrm         2/2     Running   5
(2m42s ago)   20h
linkerd-proxy-injector-5497f6fbd7-qzwvx   2/2     Running   4
(2m42s ago)   20h


yourname@ubuntu-vm:~$ kubectl get pods -n ingress-nginx

ingress-nginx-admission-create-r7gvr       0/1     Completed   0
40h
ingress-nginx-admission-patch-6prps        0/1     Completed   0
40h
ingress-nginx-controller-b66cc4b74-pp4kh   1/1     Running     4
(2m7s ago)   40h
```


6. For this lab, you will use another application provided by Buoyant (the creators of Linkerd) to demonstrate Linkerd's ability to perform retries and timeouts on individual services in the mesh. Install the booksapp application:

```
yourname@ubuntu-vm:~$ kubectl create ns booksapp && \
  curl -sL https://run.linkerd.io/booksapp.yml \
  | linkerd inject - \
  | kubectl -n booksapp apply -f -

namespace/booksapp created

service "webapp" skipped
deployment "webapp" injected
service "authors" skipped
deployment "authors" injected
service "books" skipped
deployment "books" injected
deployment "traffic" injected

service/webapp created
deployment.apps/webapp created
service/authors created
deployment.apps/authors created
service/books created
deployment.apps/books created
deployment.apps/traffic created
```

7. Next, configure the Nginx Ingress to use the new app.

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
—
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: booksapp
  name: ingress-booksapp
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/service-upstream: "true"
    consul.hashicorp.com/connect-inject: “true”
spec:
  ingressClassName: nginx
  rules:
```



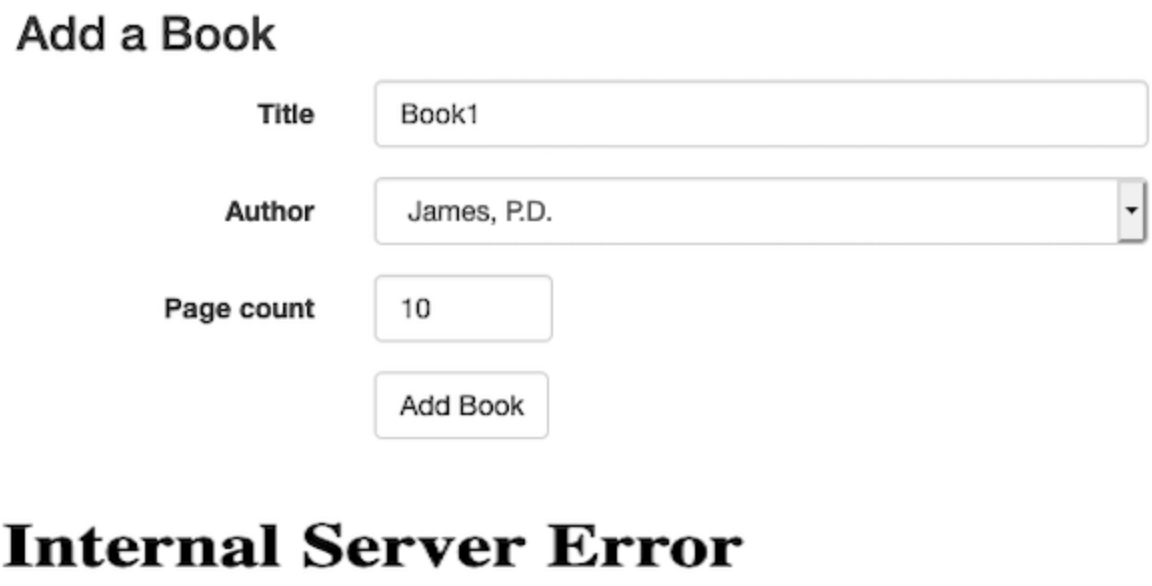
```
          - http:
              paths:
              - pathType: Prefix
                path: "/"
                backend:
                  service:
                    name: webapp
                    port:
                      number: 7000
      EOF

      ingress.networking.k8s.io/ingress-booksapp created
```

8. Create a Kubernetes **port-forward**:

```
      yourname@ubuntu-vm:~$ kubectl port-forward
service/ingress-nginx-controller -n ingress-nginx 8080:80
```

9. Navigate to 127.0.0.1:8080 in a web browser to interact with the new application. This application is a database of books and authors. Clicking a book will give you information about the book like the page count and author. Clicking an author will give you a list of the books they have written. You can also add authors and books to the database. Add several books to the database, and you will find that the function fails around half the time.



10. Linkerd exposes the [ServiceProfile] configuration to see per-route metrics, and to configure timeouts and automatic retries on those routes. This allows for you to pinpoint

some errors and mitigate them at the network level while you work on patching the code. For HTTP services, there are two mechanisms for creating **ServiceProfile**s. You can use live traffic to automatically create **ServiceProfiles** based on requests through the proxy or you can use OpenAPI (Swagger) documentation for the service to generate a **ServiceProfile**. To guarantee you have a full understanding of the available routes, you will use the available OpenAPI docs for this service.

```
yourname@ubuntu-vm:~$ curl -sL
https://run.linkerd.io/booksapp/webapp.swagger \
  | linkerd -n booksapp profile --open-api - webapp \
  | kubectl -n booksapp apply -f -

serviceprofile.linkerd.io/webapp.booksapp.svc.cluster.local
created
```

11. Wait a few minutes, then take a look at the metrics for these routes from the command line. The **linkerd routes** command returns a snapshot of the performance of each route in the **ServiceProfile** above. The output appears to indicate that **POST** requests to the **/books** endpoint fail around half the time.

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp
service/webapp

ROUTE                       SERVICE   SUCCESS     RPS
LATENCY_P50   LATENCY_P95   LATENCY_P99
GET /                       webapp    100.00%   0.6rps        15ms
      20ms        20ms
GET /authors/{id}           webapp    100.00%   0.6rps        8ms
      10ms        10ms
GET /books/{id}             webapp    100.00%   1.1rps        8ms
      18ms        20ms
POST /authors               webapp    100.00%   0.6rps        9ms
      19ms        20ms
POST /authors/{id}/delete   webapp    100.00%   0.6rps        15ms
      20ms        20ms
POST /authors/{id}/edit     webapp          -     -           -
      -           -
POST /books                 webapp     45.83%   3.0rps
9ms         18ms        20ms
POST /books/{id}/delete     webapp    100.00%   0.6rps        8ms
      10ms        10ms
```



```
POST /books/{id}/edit       webapp      42.35%    1.4rps        16ms
      84ms        97ms
[DEFAULT]                   webapp           -     -           -
      -           -
```

12. You now know that requests from the webapp service to the books service appear to be failing. You can get more visibility to this by creating a **`ServiceProfile`** for the upstream services as well. You can do this with OpenAPI documentation for the books servicel.

```
yourname@ubuntu-vm:~$ curl -sL
https://run.linkerd.io/booksapp/books.swagger \
  | linkerd -n booksapp profile --open-api - books \
  | kubectl -n booksapp apply -f -

serviceprofile.linkerd.io/books.booksapp.svc.cluster.local
created
```

13. Wait a few minutes, then run the **`linkerd routes`** command below to review the requests from the webapp service to the books service. You can see that **`POST`** *and* **`PUT`** actions seem to be the ones causing errors.

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp
deploy/webapp --to service/books

ROUTE                       SERVICE   SUCCESS      RPS
LATENCY_P50   LATENCY_P95   LATENCY_P99
DELETE /books/{id}.json     books   100.00%   1.3rps          5ms
      9ms         10ms
GET /books.json             books   100.00%   1.9rps          2ms
      3ms         4ms
GET /books/{id}.json        books   100.00%   2.5rps          2ms
      3ms         17ms
POST /books.json            books 44.25%    2.9rps              5ms
      16ms        19ms
PUT /books/{id}.json        books 54.05%    1.2rps              6ms
      84ms        97ms
[DEFAULT]                   books       -     -           -
-           -
```

14. It is a little strange that two independent routes, both the **`POST`** *and* **`PUT`**, are erroring at the same rate. You can dig a little deeper by seeing if both of these routes rely on

another service. Checking the output of **linkerd tap** with the command below will give you a log of the requests flowing through the books service. Hit **Control-C** to stop the log from scrolling so you can read it. From that log, you should find that the books service is making a lot of **HEAD** requests to the authors service. Based on the number of 503s that are being returned, those requests appear to be failing quite frequently.

```
yourname@ubuntu-vm:~$ linkerd viz tap -n booksapp deploy/books

req id=9:49 proxy=out src=10.244.0.53:37820 dst=10.244.0.50:7001
tls=true :method=HEAD :authority=authors:7001
:path=/authors/3252.json
rsp id=9:49 proxy=out src=10.244.0.53:37820 dst=10.244.0.50:7001
tls=true :status=503 latency=2197µs
end id=9:49 proxy=out src=10.244.0.53:37820 dst=10.244.0.50:7001
tls=true duration=16µs response-length=0B
```

15. Let's investigate this further. To start, create a **ServiceProfile** for the authors service from the swagger doc.

```
yourname@ubuntu-vm:~$ curl -sL
https://run.linkerd.io/booksapp/authors.swagger \
  | linkerd -n booksapp profile --open-api - authors \
  | kubectl -n booksapp apply -f -

serviceprofile.linkerd.io/authors.booksapp.svc.cluster.local
created
```

16. After creating the **ServiceProfile**, wait a few minutes and then check **linkerd routes** for requests from the books service to the authors service. You will see that the **HEAD** requests are failing.

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp deploy/books
--to service/authors

ROUTE                       SERVICE   SUCCESS     RPS
LATENCY_P50   LATENCY_P95   LATENCY_P99
DELETE /authors/{id}.json   authors         -     -          -
      -           -
GET /authors.json           authors         -     -          -
      -           -
GET /authors/{id}.json      authors         -     -          -
      -           -
```



```
HEAD /authors/{id}.json     authors     54.87%   3.8rps
1ms         2ms         3ms
POST /authors.json          authors          -     -          -
      -           -
[DEFAULT]                   authors          -     -          -
      -           -
```

17. From the output of the commands in the previous steps, you can see that linkerd is reporting similar behavior to what we have been seeing while interacting with the application. The actions that add books to the database fail around half the time. Based on your debugging work, it seems that this is actually due to an issue with the **`HEAD`** requests to the authors service. This issue needs to be remedied quickly. Since coding can be a time consuming activity, Linkerd gives you the ability to easily put some mitigations in place while you work on debugging what went wrong.

    a. The **`ServiceProfile`** is not just capable of *reporting* per-route metrics; it can also *configure* per-route rules for linkerd. Based on the data above, adding books to the database appears to be failing around half the time. You can configure automatic retries on those actions so that Linkerd will try to re-send the request instead of reporting an error to your users. Manually edit the **`ServiceProfile`** by running **`kubectl edit`** to open it in a text editor.

    ```
    yourname@ubuntu-vm:~$ kubectl edit -n booksapp
    serviceprofile authors.booksapp.svc.cluster.local

    [Opens a text editor on success]
    ```

    b. Examine the **`ServiceProfile`** and find the **`HEAD /authors/{id}.json`** routes. Adding **`isRetryable: true`** to that configuration will tell Linkerd that it should retry requests to that route. Edit the **`ServiceProfile`** so it matches the **`isRetryable: true`** configuration below. Save and exit the editor to apply the edit.

    ```
    ...
      - condition:
          method: HEAD
          pathRegex: /authors/[^/]*\.json
        isRetryable: true
        name: HEAD /authors/{id}.json

    serviceprofile.linkerd.io/authors.booksapp.svc.cluster.local
    edited
    ```


18. You have now configured retries for **HEAD** requests to **/authors/{id}.json** in the **authors** service. After giving it some time, check **linkerd routes** to confirm that the success rate has now climbed back up to 100%.

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp
service/webapp

ROUTE                       SERVICE   SUCCESS     RPS
LATENCY_P50   LATENCY_P95   LATENCY_P99
GET /                       webapp    100.00%   0.9rps         16ms
      25ms         29ms
GET /authors/{id}           webapp    100.00%   0.9rps         8ms
      15ms         19ms
GET /books/{id}             webapp    100.00%   1.7rps         8ms
      10ms         10ms
POST /authors               webapp    100.00%   0.9rps         13ms
      19ms         20ms
POST /authors/{id}/delete   webapp    100.00%   0.9rps         18ms
      29ms         30ms
POST /authors/{id}/edit     webapp          -     -            -
      -            -
POST /books                 webapp    100.00%   1.7rps         15ms
      20ms         28ms
POST /books/{id}/delete     webapp    100.00%   0.9rps         8ms
      10ms         10ms
POST /books/{id}/edit       webapp    100.00%   0.9rps         18ms
      72ms         94ms
[DEFAULT]                   webapp          -     -            -
      -            -
```

19. Congratulations! You have successfully mitigated an issue for your users without having to make any code changes. Your team now has time to debug and find the right changes to make, and does not have to rush to push out a bug fix!

20. While this is exciting for your users, digging in a little deeper will show you the effect it is having on your system. If you pass the **-o wide** flag to **linkerd routes**, it will show you that the **EFFECTIVE_SUCCESS** and **ACTUAL_SUCCESS** rates are quite different. In the sample output below, **EFFECTIVE_SUCCESS** is 100.00% but **ACTUAL_SUCCESS** is 48.14%. This is because the Linkerd sidecar proxy in the books service is now sending more requests to the authors service when a request fails. You can see this in the

EFFECTIVE_RPS and **ACTUAL_RPS** values (2.6rps and 5.4 rps, respectively, in the sample output).

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp deploy/books
--to service/authors -o wide

ROUTE                        SERVICE   EFFECTIVE_SUCCESS
EFFECTIVE_RPS   ACTUAL_SUCCESS   ACTUAL_RPS   LATENCY_P50
LATENCY_P95   LATENCY_P99
DELETE /authors/{id}.json    authors                    -
      -                 -           -           -           -
-
GET /authors.json            authors                    -
      -                 -           -           -           -
-
GET /authors/{id}.json       authors                    -
      -                 -           -           -           -
-
HEAD /authors/{id}.json      authors           100.00%
2.6rps           48.14%      5.4rps            3ms         10ms
17ms
POST /authors.json           authors                    -
      -                 -           -           -           -
-
[DEFAULT]                    authors                    -
      -                 -           -           -           -
-
```

21. The other effect this has had is it slightly increases the latency of adding books. Running **linkerd routes** will show that there is a relatively high P95 latency.

```
yourname@ubuntu-vm:~$ linkerd viz routes -n booksapp
deploy/webapp --to svc/books

ROUTE                        SERVICE   SUCCESS     RPS
LATENCY_P50   LATENCY_P95   LATENCY_P99
DELETE /books/{id}.json      books   100.00%   0.9rps         8ms
      10ms         10ms
GET /books.json              books   100.00%   1.8rps         3ms
      5ms          5ms
GET /books/{id}.json         books   100.00%   2.6rps         2ms
      3ms          3ms
```



```
POST /books.json            books   100.00%   1.6rps          14ms
      19ms        20ms
PUT /books/{id}.json        books   100.00%   0.8rps          16ms
      82ms        97ms
[DEFAULT]                   books      -      -              -
-           -
```

22. This issue, like retries, can be mitigated by Linkerd until your team can fully resolve the issue.

    a. Run **kubectl edit** to open the **ServiceProfile** in a text editor.

    ```
    yourname@ubuntu-vm:~$ kubectl edit -n booksapp
    serviceprofile books.booksapp.svc.cluster.local

    [Opens a text editor on success]
    ```

    b. Find the **PUT** route you need to configure retries on, and add the **timeout: 25ms** configuration below. This will cause attempts to time out more quickly. Save and exit the editor to apply the edit.

    ```
    ...
      - condition:
          method: PUT
          pathRegex: /books/[^/]*\.json
        name: PUT /books/{id}.json
        timeout: 25ms

    serviceprofile.linkerd.io/books.booksapp.svc.cluster.local
    edited
    ```

23. Wait a few minutes and run **linkerd routes** again to see the latest data. The latency numbers include time spent in the webapp itself, so they will remain above the 25 ms threshold you set. You can see that the timeouts are working because the **EFFECTIVE_SUCCESS** rate drops as the linkerd proxy times out on some requests. In the example output below, that rate is 92.16%. There are always tradeoffs when mitigating issues with microservices, and you must decide if setting around a 90% success rate with moderate latency is a reasonable way to leave the system while your team works on a fix for the service.

    ```
    yourname@ubuntu-vm:~$ linkerd -n booksapp viz routes
    deploy/webapp --to svc/books -o wide
    ```



```
ROUTE                        SERVICE    EFFECTIVE_SUCCESS
EFFECTIVE_RPS   ACTUAL_SUCCESS   ACTUAL_RPS   LATENCY_P50
LATENCY_P95   LATENCY_P99
DELETE /books/{id}.json      books            100.00%
0.9rps          100.00%    0.9rps           8ms         10ms
10ms
GET /books.json              books            100.00%
1.9rps          100.00%    1.9rps           3ms         4ms
5ms
GET /books/{id}.json         books            100.00%
2.7rps          100.00%    2.7rps           2ms         3ms
3ms
POST /books.json             books            100.00%
1.6rps          100.00%    1.6rps           14ms        19ms
20ms
PUT /books/{id}.json         books            92.16%
0.8rps          100.00%    0.8rps           11ms        77ms
96ms
[DEFAULT]                    books                -
-               -          -          -         -           -
```

24. You've reached the end of this lab. You will continue to use the Linkerd cluster in your next lab, so leave it running if you plan to move on to the next lab now. Otherwise, stop the Linkerd cluster by running this command:

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

8af4c08524df
```

