CROSS LAYER TRACING IN KUBERNETES

Runzhou Han

Aditya Chechani

Reet Chowdhary

Taiga: https://tree.taiga.io/project/msdisme-2018-bucs528-template-7/

GitHub: https://github.com/BU-NU-CLOUD-SP18/Cross-Layer-Tracing-in-Kubernetes

Tasks Completed

- Install Kubernetes on local cluster
  - We were not able to do this on the MOC after spending 6+ hours on it with our mentor
- Write a Node.js application, containerize it, deploy it in Minikube and write an ingress to expose it to external traffic
- Install nginx on one of the local machines
- Use nginx to set up proxy server routed to two separate flask apps
- Find section of nginx source code pertaining to manipulation of HTTP headers

Ingress and Nginx

- An Ingress is a Kubernetes resource that lets you configure an HTTP load balancer for your Kubernetes services.
- The load balancer exposes your services to clients outside of your Kubernetes cluster.
  - Could be done by a custom URL (Service A at /serviceA and Service B at /serviceB)
  - Could be done by multiple hostnames (serviceA.example.com or serviceB.example.com)

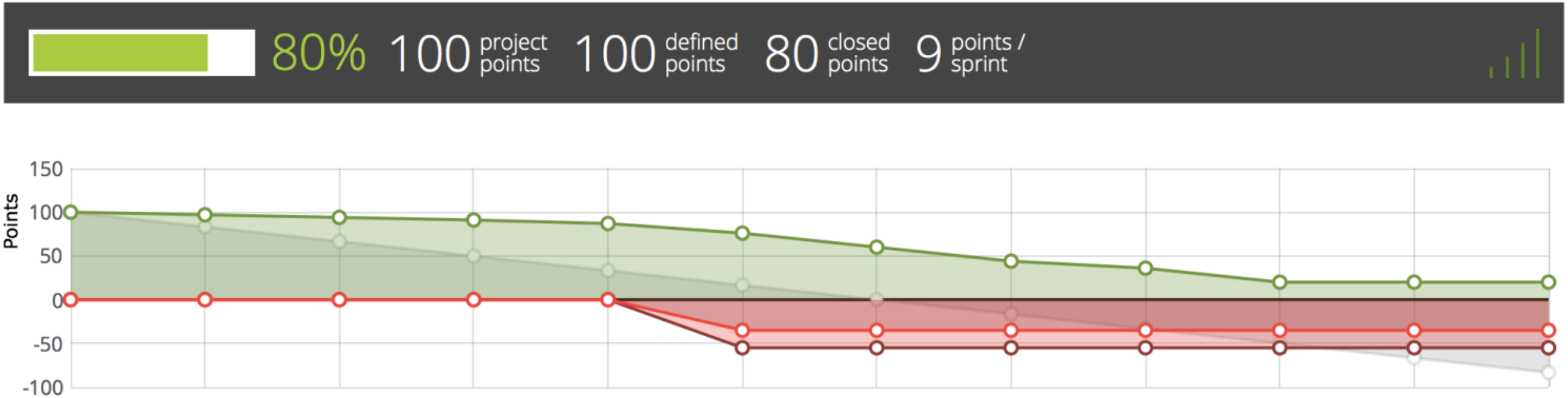
Our findings on looking at Nginx and Kubernetes source code

- Nginx can be used as a load balancer
  - Requests come in from the outside world and nginx routes the requests to the services inside the cluster
- The ingress controller looks at the available ingress resources using the Kubernetes API and makes changes to the configuration of the load balancer (nginx in this case)

Nginx source code

```
12
13  static void ngx_http_wait_request_handler(ngx_event_t *ev);
14  static void ngx_http_process_request_line(ngx_event_t *rev);
15  static void ngx_http_process_request_headers(ngx_event_t *rev);
16  static ssize_t ngx_http_read_request_header(ngx_http_request_t *r);
17  static ngx_int_t ngx_http_alloc_large_header_buffer(ngx_http_request_t *r,
18      ngx_uint_t request_line);
19
20  static ngx_int_t ngx_http_process_header_line(ngx_http_request_t *r,
21      ngx_table_elt_t *h, ngx_uint_t offset);
22  static ngx_int_t ngx_http_process_unique_header_line(ngx_http_request_t *r,
23      ngx_table_elt_t *h, ngx_uint_t offset);
24  static ngx_int_t ngx_http_process_multi_header_lines(ngx_http_request_t *r,
25      ngx_table_elt_t *h, ngx_uint_t offset);
26  static ngx_int_t ngx_http_process_host(ngx_http_request_t *r,
27      ngx_table_elt_t *h, ngx_uint_t offset);
28  static ngx_int_t ngx_http_process_connection(ngx_http_request_t *r,
29      ngx_table_elt_t *h, ngx_uint_t offset);
30  static ngx_int_t ngx_http_process_user_agent(ngx_http_request_t *r,
31      ngx_table_elt_t *h, ngx_uint_t offset);
32
```

Taiga burndown chart



Next steps

Install Nginx from source and set it up with our two applications as backend.

Add some logs to the part of source code we think are responsible for header manipulation.

Figure out exactly where the header manipulation is being done.

Add trace points to these parts and make a docker image of our customized Nginx(This is because when building kubernetes from source it pulls the image of Nginx from docker and builds it)