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

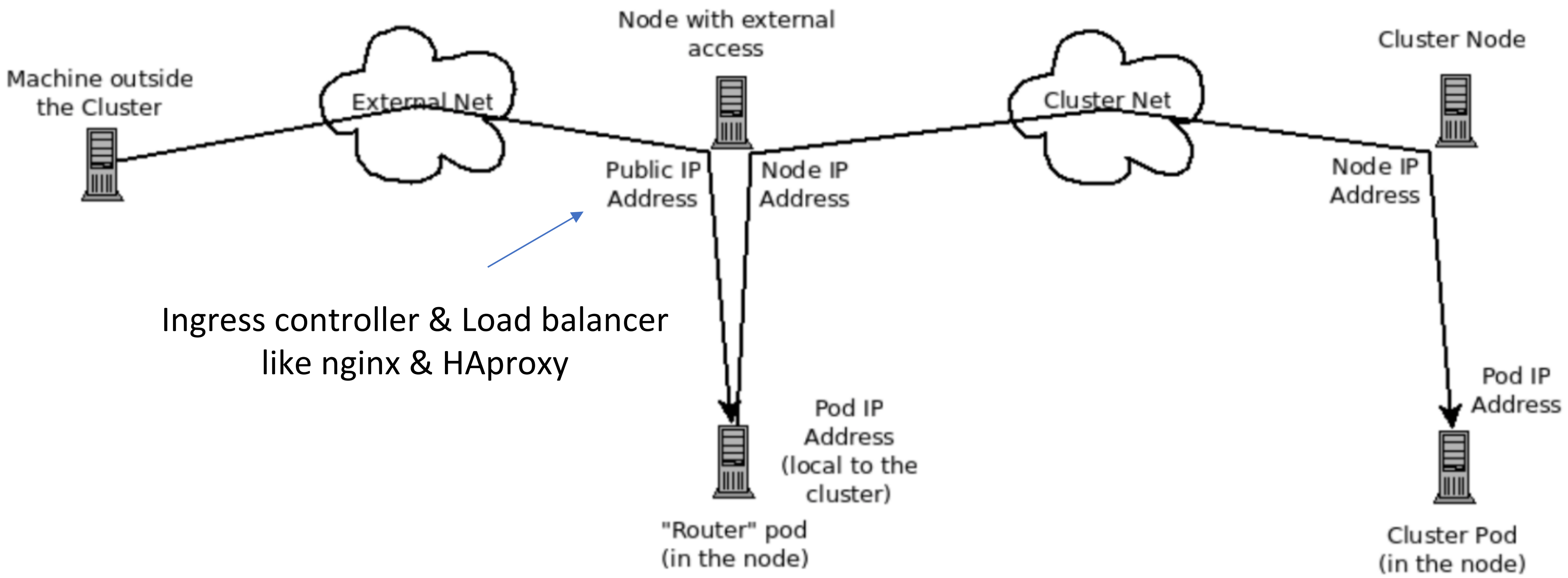
Review

- Kubernetes
    - Ingress
        - Ingress controller
            - Nginx
            - Haproxy
    - Service
- Tracing
    - Data plane

Task Completed

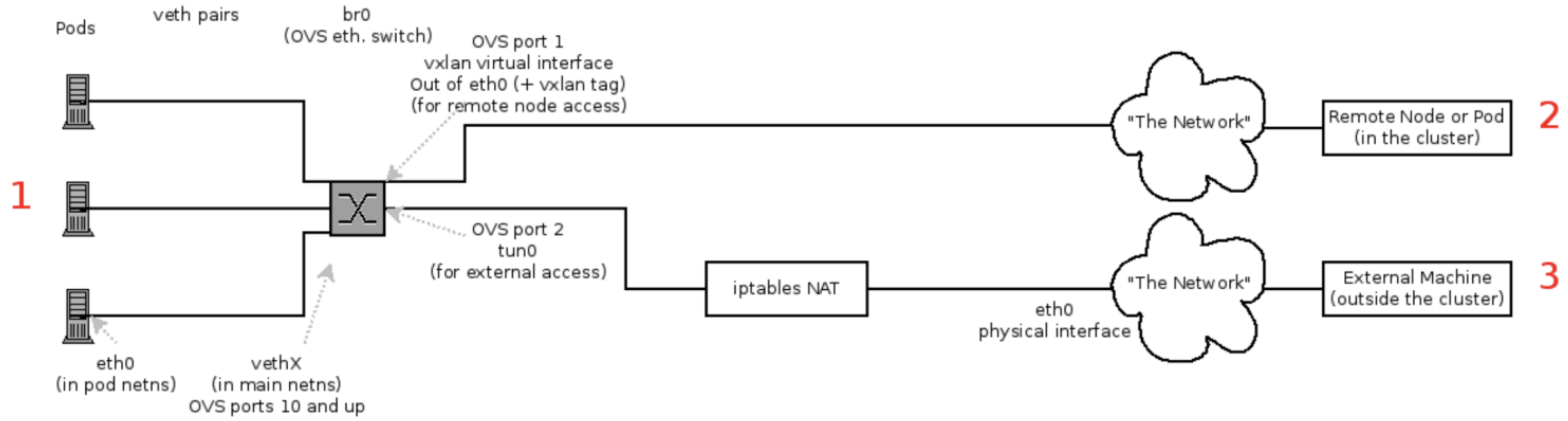
1. Setup MOC
2. Deploy Kubernetes
3. Trace HotRod using Jaeger on MOC and locally
4. Get logs from Kubernetes Control Plane and make sense of it
5. Create a flask application and deploy in Kubernetes
6. Get Nginx Logs locally

Kubernetes Data Plane


Machine outside the Cluster
External Net
Node with external access
Public IP Address
Node IP Address
Cluster Net
Cluster Node
Node IP Address
Pod IP Address (local to the cluster)
"Router" pod (in the node)
Pod IP Address
Cluster Pod (in the node)

Ingress controller & Load balancer like nginx & HAproxy

Kubernetes Data Plane



Ingress

- Ingress is a collection of rules that allow inbound connections to reach the internal cluster services
- In order for ingress to work the cluster must have an Ingress controller running
- Currently GCE and nginx are supported, some instances of the documentation indicates that HAproxy is the load balancer
- The plan is to add trace points to nginx or HAproxy in the next sprint in order to trace ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```

When you create the Ingress with `kubectl create -f`:

```
$ kubectl get ing
NAME      RULE          BACKEND   ADDRESS
test      -
          foo.bar.com
          /foo          s1:80
          /bar          s2:80
```

Containerizing and Deploying Flask

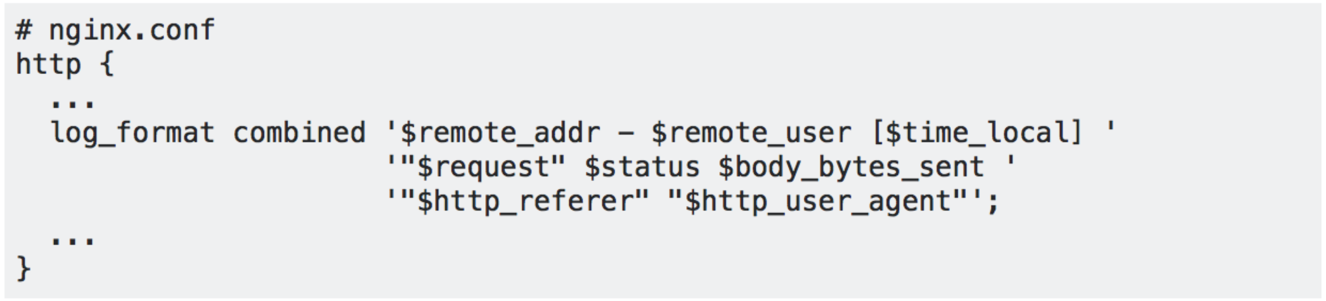
- In order to display the functionality of Kubernetes Ingress features we decided to use a Flask app
- The application returns a simple HTML response upon receiving a GET request
- We were able to containerize the application and deploy it and receive a response from it on the local network
- We must now expose the application to the outside world and send the requests through the ingress

Change in Plan:

Deploy a simple containerized flask application in Kubernetes

Establish communication from outside world

Get logs/traces from the Ingress Controller (Nginx).



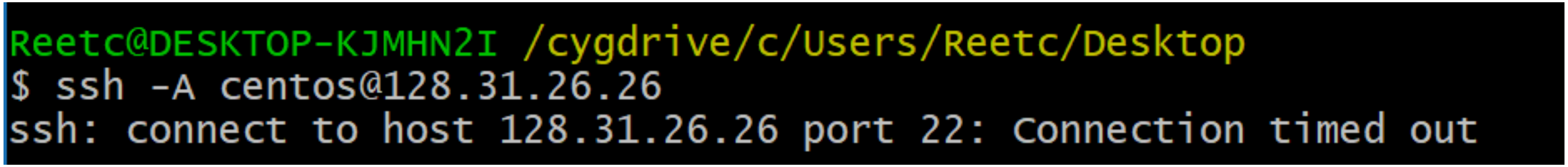
```
# nginx.conf
http {
  ...
  log_format combined '$remote_addr - $remote_user [$time_local] '
                      '"$request" $status $body_bytes_sent '
                      '"$http_referer" "$http_user_agent"';
  ...
}
```

Figure out where the trace points can be added in Nginx

If time permits, look into OVS switch which is responsible for communication between nodes inside Kubernetes.

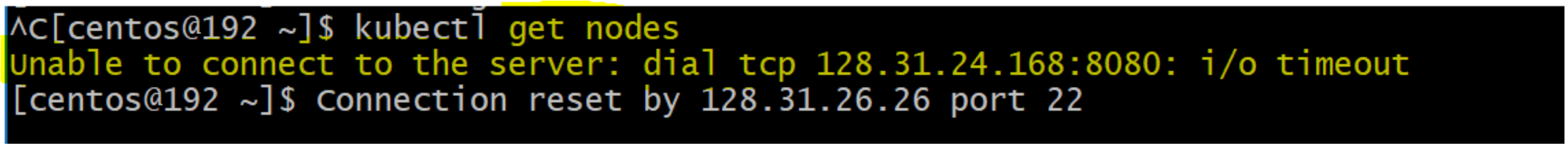
Some Issues

Frequent Connection issues



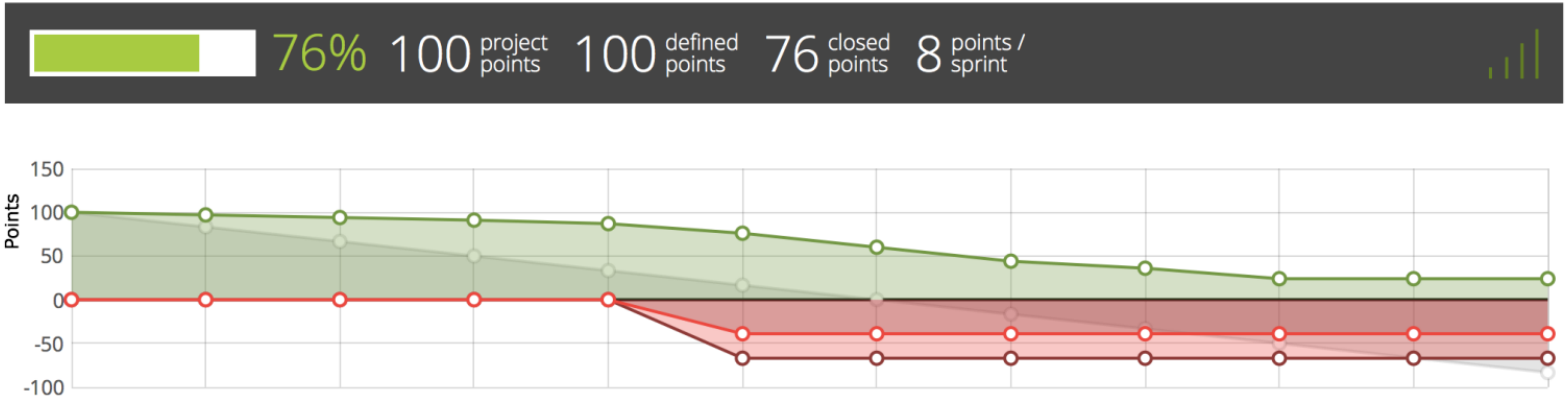
```
Reetc@DESKTOP-KJMHN2I /cygdrive/c/Users/Reetc/Desktop
$ ssh -A centos@128.31.26.26
ssh: connect to host 128.31.26.26 port 22: Connection timed out
```

Recently kubectl is timing out after we’re already on the master node



```
^C[centos@192 ~]$ kubectl get nodes
Unable to connect to the server: dial tcp 128.31.24.168:8080: i/o timeout
[centos@192 ~]$ Connection reset by 128.31.26.26 port 22
```

Taiga Burndown chart



Release Planning

- In sprint 5, we will:
  - Add tracing points to Kubernetes ingress controller
  - Show that we can collect tracing events of Kubernetes
- In final sprint, we will:
  - Compare Kubernetes tracing and application tracing

Questions?