Dmitri Lerko

Personal blog where I cover my experiences and best practices with GCP, GKE, GitOps, Certifications and DevOps.

Kubernetes Networking Problems Due to the Conntrack

Feb 15, 2020
4 min read
gcp gke

TL;DR

Kubernetes nodes set **conntrack_max** value proportionally to the size of the RAM on the node. High load applications (especially on small nodes) can easily exceed **conntrack_max** and result in connection resets and timeouts. There is several options to deal with this issue.

Theory

conntrack is a feature built on top of Netlifier framework. It is essential for performant complex networking of Kubernetes where nodes need to track connection information between thousands of pods and services.

Wikipedia's definition:

> Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relate all of the packets which may make up that connection. NAT relies on this information to translate all related packets in the same way, and iptables can use this information to act as a stateful firewall.

conntrack on Linux systems is not unbound, in Kubernetes default value can be found in the **node_nf_conntrack_entries_limit** prometheus metrics (requires node_exporter) or via:

```
sysctl net.ipv4.netfilter.ip_conntrack_max
```

It is suggested that default conntrack_max is limited to 65k entries and can be calculated with the following formula:

```
CONNTRACK_MAX = RAMSIZE (in bytes) / 16384 / (x / 32) where x is the number of bits in a pointer (for example, 32 or 64 b
```

Above calculation indicates that **conntrack_max** value is directly proportional to the node's memory. This is not what we've observed with GKE nodes. Our observation shows that for each MB of memory, we get roughly 5 **conntrack_max**.

Setup

At loveholidays we are running some high load services, one of them is an aggregation service which acts as a proxy between two other services (each consisting of 100+ instances). This service continuously operates at 10'000+ requests per second and can spike up to x4 of its baseload. This means that our aggregation proxy service needs to continuously track a large number of connections.

Incident

Shortly after the change in service architecture, node pool sizing (we've halved RAM on the nodes running the aggregation service) and sharding of downstream service (doubled number of connections that need to be tracked) we've started noticing various connectivity problems with the service and elevated error rates.

> Caused by: org.apache.http.NoHttpResponseException: xxxxx failed to respond

HAProxy reported high connection reset errors that kept climbing up. We've started analysing available networking metrics and also stumbled across this article. We've used two following prometheus metrics to understand our current usage *node_nf_conntrack_entries* and *node_nf_conntrack_entries_limit*. Once we overlayed our current entries with entry limit it was clear that some of our nodes are running out of conntrack tablespace:

```
# This query displays % of utilisation of conntrack table space
(node_nf_conntrack_entries / on (pod) node_nf_conntrack_entries_limit / on (pod) group_right  kube_pod_info) * 100
```

Solutions

Immediate fix

Knowing that conntrack_max is simply a ratio to the size of RAM on the node - we've chosen to schedule our application on nodes with larger nodes, while also limiting how many pods of aggregation service we are running per node. This has resolved the immediate problem.

Kubernetes solution

Kubernetes version 1.15 or newer should have the following fix in place https://github.com/kubernetes/kubernetes/pull/74840

Monitoring and alerting

We are now monitoring and alert on high-conntrack utilisation using this simple prometheus query:

```
(node_nf_conntrack_entries / on (pod) node_nf_conntrack_entries_limit / on (pod) group_right  kube_pod_info) > 0.75
```

Init-container or DaemonSet

In situations where your service runs on dedicated nodes, it is possible to allow your application to manipulate node's **conntrack_max** via **sysctl** command.

This approach has numerous downsides:

- Your application has no awareness of node's default **conntrack_max**, so if nodes RAM is going to get increased, it is possible that below scripts will reduce **conntrack_max**. While a more complex script can be developed, it does not feel right for application to manage infrastructure as it has a completely different primary purpose.
- Multiple applications with init-scripts can result in changing **contrack_max** values as pods get deployed on the nodes. This can be tricky to debug in scenarios where pods are frequently recreated due to releases or horizontal pod autoscaler changes.
- Increase of **conntrack_max** may result in increased RAM usage on the node and possible increase in latency as there is more to process per each connection.

Init-container

```
# Init-container which will run on pod startup
initContainers:
- name: sysctl
  image: marketplace.gcr.io/google/centos7
  args:
  - -c
  - sysctl -w net.netfilter.nf_conntrack_max=262144
  command:
  - /bin/sh
  securityContext:
    privileged: true
```

DaemonSet

```
# Daemon-set which will run on node-startup on node matching on nodekey: nodevalue.
# Sleep is a hacky way to prevent DaemonSet termination.
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: sysctl-conntrack
spec:
  template:
    spec:
      nodeSelector:
        nodekey: nodevalue
      containers:
      - name: sysctl-buddy
        image: marketplace.gcr.io/google/centos7
        securityContext:
          privileged: true
        command: ["/bin/sh"]
        args:
          - -c
          - sysctl -w net.netfilter.nf_conntrack_max=262144 && sleep 31536000000
```

Further reading

- kube-proxy Subtleties: Debugging an Intermittent Connection Reset
- When Linux conntrack is no longer your friend
- SOLVING CONNECTION RESET ISSUE IN KUBERNETES
- Chasing a Kubernetes connection reset issue
- A reason for unexplained connection timeouts on Kubernetes/Docker

---

Find posts by tags

certifications dataflow flux gcp gcr gitops gke jekyll kubernetes label_replace prometheus recording-rules redis sensu