骨幹 Netflow 即時轉入大資料平台之方法設計與實作

梁明章
國家高速網路與計算中心
liangmc@narlabs.org.tw

摘要

本文將說明我們如何設計並實作四階段分散架構來處理 TWAREN 骨幹持續產生的 Netflow 大資料流使之順利即時轉換進入 ElasticSearch 大資料平台，並且使用盡量少的伺服器就能達成任務，同時也說明如何開發自動程式監控大資料平台，當 ElasticSearch 平台資源不足以負擔網路大規模異常暴增的 Netflow 資料流發生崩潰時，監控程式可以自動進行應急處置而使平台盡快恢復服務。

關鍵詞：ElasticSearch、Netflow、骨幹網路、大資料平台、自動監控、分散處理。

1. 前言

TWAREN NOC 維運已有十多年，除了致力於開發常規網管系統之外，我們還努力開發骨幹級的網路行為異常偵測系統，盡量達到即時偵測、快速通報、快速處置，甚至是自動處置，因為自動處置肯定比呼叫值班工程師上線處理要快得多，而「異常行為即時自動偵測」的重要基礎之一，就是全骨幹 Netflow 資料可以即時統計、分析與查詢，由於骨幹 Netflow 不僅資料量龐大，而且是屬於持續不斷產生的大資料，光是如何讓資料流即時持續進入大資料平台就得花一番心思。

TWAREN 研究網路目前有13個 GigaPOP(類似區網中心)與4個核心節點(台北、新竹、台中、台南)，每 GigaPOP 有一個主路由器，每核心節點有兩個主路由器跟一個邊界路由器，如下錯誤! 找不到參照來源。，以及在美國境內租用三個邊界路由器，如下錯誤! 找不到參照來源。，全網總共28個大型路由器，全部都設定發送1:1無取樣比率的Netflow，凡是經過TWAREN網域的傳輸都會留下紀錄，日常產生的 Netflow 筆數約在每秒四萬筆到八萬筆之間，若發生大規模 DDoS 之類攻擊擴散到很多學校時，產生的筆數會更多，由於 NOC 需要利用 Netflow 的即時資訊特性來快速判斷網路當下是否有發生大規模攻擊好儘快反應處置，因此 TWAREN 大資料平台匯入 Netflow 只能採取持續匯入及短時區間結算統計(目前採用五分鐘週期)的方式盡量做到快速反應，因此五分鐘的大資料必須要在五分鐘內做完所有運算、判斷、告警、追蹤等程序，否則就會滾雪球般往後堆積延遲，直到系統崩潰或失去快速處置的意義。

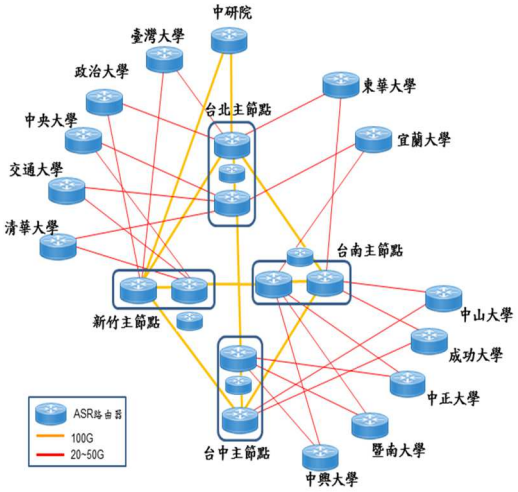

圖 1 TWAREN 國內路由器圖

圖 2 TWAREN 國外路由器圖

因為 TWAREN 的 Netflow 有收納後立即可被異常判斷系統查詢的需求，因此我們選擇 ElasticSearch Stack**[1]**作為大資料平台(後文簡稱 ES)，因為 ES 的特性就是會把每秒鐘的資料作成一棵索引樹，因此進入 ES 的新資料在一秒後就可以查詢，而 ES 的 Logstash 軟體雖然能轉換 Netflow 成 JSON 匯給 ES，但缺點就是轉換效率太差，我們最初測試時 TWAREN 還在前一代設備(Cisco N7k 系列)，Netflow 使用第五版(格式標準參考**[2]**，無法包容 IPv6)，Netflow Version 5是固定欄位格式、數量與長度的，因此路由器可以利用硬體晶片產生 Netflow，而網管這邊用 Structure Array 方式處理 Netflow 的演算需求也較低，然而當時我們實測 Logstash 在一台中階伺服器上全力運行也只能達到每秒七千至九千筆轉換速率，想平穩吃下 TWAREN 的 Netflow 需要將近十台實體伺服器，我們無法接受這樣的耗費，更何況其後 TWAREN 更新至100G 設備(Cisco ASR9k 系列)，新設備只能產生 Netflow 第九版，格式基本標準參考 Cisco IOS NetFlow Version 9 Flow-Record Format**[3]**，擴充標準則參考 IPFIX (IP Flow Information Export**[4]**)，Netflow Version 9 不只可以包含 IPv6，還能包含 VLAN、MPLS 等等多種資訊，因此欄位數量、格

式全都是隨時可變化的，因為彈性又複雜，Netflow 程式面臨處理運算能力需求大增的困擾，Netflow Version 9 的彈性變化特性使處理程式必須逐一分析判斷每個欄位格式與長度，我們實測 Logstash 只能達到每秒兩三千筆的效率，我們不可能花十幾台伺服器來跑 Logstash，因此我們自己開發高效率需求資源低且可分散運行的轉換程式，需求的伺服器越少越好，本文將會介紹我們使用哪些方法來達成將每秒十萬筆的 Netflow 持續匯入 ES 的目標(我們 ES 即時處理的 Netflow 不是只有 TWAREN 骨幹的)。

TWAREN 從2014年開始嘗試幾種大資料平台收容網路設備與伺服器的 Syslog、SNMP Trap 與 MIB、Web access log、Netflow，後來主要因為即時異常判斷的需求選擇了 ElasticSearch，從1.x 版一路使用到現在7.x 版，這六年間也累積了一些管理經驗心得，本文主要內容第二部分將會說明我們如何自動監控 ES Cluster 並自動處理 Cluster 異常狀態的作法，提供給有類似情況的管理者做為參考。

2. 分散轉換 Netflow 進 ES 的方法與實作

想將 Netflow 資料匯入 ES，先要將 Netflow 資料轉換成 JSON 文檔，主要問題在於轉換的效率，以我們的需求，每秒十萬筆，肯定無法利用單一程序來達成，因此分散同時運行勢在必行，下列說明我們的實現方法，架構如下錯誤! 找不到參照來源。，並分小節說明於後。

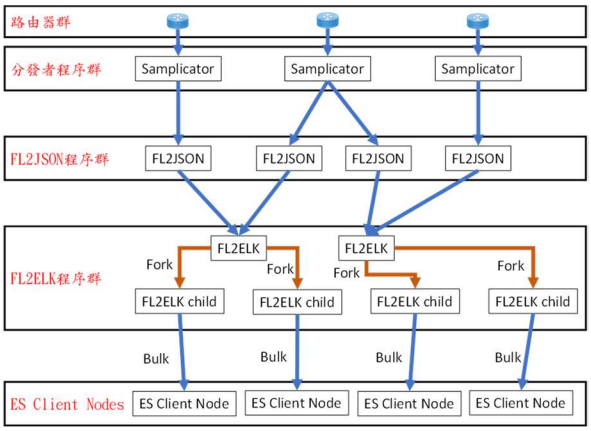

圖 3 Netflow 轉換匯入之分散架構圖

2.1 分發與分散 Netflow 封包

首先我們需要注意一項常被忽略的點，網路程式想接收資料，都必須先跟系統請求 Socket 資源，然後系統會為此 Socket 配置輸出入雙向緩衝佇列(in/out queue)，此佇列長度一般稱為 Backlog Length，然後程式再向系統請求一個 Port 號碼將此 Socket 資源繫結(BIND)上去，之後系統若發現網路上進來的封包目的是該 Port 就會把封包放入該 Socket 的輸入佇列，程式必須去輸入佇列取出封包做處理，處理完再到佇列拿取資料，如此循環，如果程式消化佇列的速度慢於網路封包進入佇列的速度，佇列將會滿溢，新來的封包就會被系統拋棄，雖然佇列長度可以調高，但如果佇列消化速度一直低於新進速度，不論佇列拉多長最後都會滿溢，因此我們一開始就實作分散方式，讓各路由器將 Netflow 送到分發者機器不同的 Port，分發者在每個 Port 都執行一隻分發程式，該程式只是單純把封包收下並直接拷貝多份後轉發給下一關，目前我們使用開源的 Samplicator**[5]**軟體做分發程式，因為該程式可以在拷貝多份轉發時重製 IP/UDP Header 來保持原始發送者(路由器)的 IP 作為封包來源 IP，這一點很重要，後續關卡都需要靠這個來源 IP 判斷這是哪一台路由器發的 Netflow。而且因為程式開源，我們可以修改 Samplicator 的原始碼，加上 Round-Bobin 平均分配派送的功能用來分散 Netflow 超級多的單一來源給多個下一關(多組 IP+Port)共同負擔。之所以第一關就只單純分發主要是因為我們 Netflow 資料不只是進入 ES 大資料平台，還要複製給專門做長期保存的系統，可以保存很多年的 Netflow 原始資料，以及複製給專門做日/月/年報表的系統。因為 Samplicator 全程只在記憶體中收封包、重製 Header、發送給多個目標等簡單動作，因此消化輸入佇列的速度不會低於新進佇列的速度，不用擔心佇列滿溢掉包的問題。

2.2 分析 Netflow 轉換成 JSON 文檔

這一關程式我們稱呼為 FL2JSON，我們以 C 語言自行開發，在 Netflow version 5 的時候因為固定欄位與長度，每個 Netflow 封包最多只能包容31筆 Netflow，因此我們的程式只需要將封包放入31筆 Structure 的陣列就能獲取封包內每一筆 Netflow 的各個欄位與值，然後將每筆 Netflow 轉印成一篇 JSON 文檔，順便還能把來源及目的 IP 利用 Geo 地理資料庫查出所屬國家、城市、組織、經緯度等資訊一起轉印到 JSON 中，然後再將 JSON 文檔以 UDP 封包用 Round-Robin 方式發送給下一關 FL2ELK Daemon，我們實作經驗，在 Netflow version 5 的時候，一個路由器 Netflow 由一隻 FL2JOSN 程序處理，並且這些 FL2JSON 都在同一台伺服器執行，佇列消化速度仍可以快過佇列堆積速度。

而到了 Netflow Version 9 的時候就複雜了，程式必須逐一解析封包內容，解析 Template，並依據 Template 所揭示的後續欄位 ID 逐一按照 Netflow 標準定義查表獲取欄位名稱與長度來拆解欄位值，Cisco Netflow Version 9 定義的欄位 ID 編號到104號，而 IPFIX 更是將定義擴充為2-Bytes，目前已定義到491號(後面還有些零星定義的欄位我們用不

到)，所以我們先得把所有欄位定義做成一個具有512筆結構的陣列，FL2JSON程式直接用欄位ID做為陣列索引直接讀取該 ID 代表的欄位名稱與欄位長度，這樣可避免多迴圈的循序尋找，此結構陣列所費記憶體不多，因此此法利多於弊。而FL2JSON 程式一邊解析欄位與值一邊列印組裝JSON，讀完一筆 Netflow 時就完成一篇 JSON 文檔，此時就可發送給下一關，依然採用 UDP 發送(因為用 TCP 需要等待回應)。

依照我們的實作經驗，針對 Netflow Version 9 將所有 FL2JSON 放在同一台中階伺服器執行在日常狀態下佇列消化會快於佇列堆積，但有些流量較大的路由器會接近臨界值，可用分發者將之平均分配給二至三隻 FL2JSON 程序來處理，如此可以避免在此關漏失封包。如果單台主機不堪負荷，可分散至兩台主機，但目前為止單台依然夠用，表示我們開發的程式有滿足使用盡量少伺服器資源的要求。

2.3 將 JSON 文檔匯入 ES

我們稱呼此階段程式為 FL2ELK，同樣是每一隻程序繫結一個 Port 接收 FL2JSON 送來的 UDP 封包，取出 JSON 文件後存入暫存檔，當暫存檔內累積文件數達到設定值時就複製(Fork)出子程序將此暫存檔送去 ES Client Node，母程序則將後續新進的 JSON 存入新的暫存檔。

每個暫存檔的累積 JSON 文件數基本上是按照 ES 的 Bulk 功能來推算，在此稍微說明 ES 的 Bulk 功能，因為 ES 匯入資料的方式是透過 RESTful API，如果每筆 JSON 文件都要一次 TCP 連線加上 HTTP 必要的多行表頭會形成很大的浪費，因此 ES 官方建議大批資料匯入應將許多 JSON 文件打一包以 ES 的 Bulk API(也是 RESTful)匯入，理論上 Bulk 包越大越省傳輸，但實際上不是越大越好，考量到 HTTP 傳輸大檔案的機制欠佳，更重要的是 ES 端收到 Bulk 包展開成 Document 物件陣列時會大量膨脹佔用有限的 JVM 記憶體空間，而且在完成整個 Bulk 請求之前無法釋放佔用的空間，此時若 JVM 記憶體空間周轉不過來就會造成 Garbage Collection Frozen(後文簡稱 GC 凍結)使 JVM 暫停服務，因此實際上 Bulk 包並非越大越好，需視實際資料量匯入速率與 JVM 處理能力來評估 Bulk 包大小。雖然網路上有些達人會作實驗並總結適當的 Bulk 大小，但由於各自的資料型態、資料產生速率、機器效能等等多項差異，達人們的結果對我們並沒多少參考價值，終究還是需要自行實驗過，簡單來說，就是逐次增加 Bulk 包的 JSON 文件數，直到 ES Client Node 發生 JVM 凍結，然後再降低文件數到安全為止，目前我們採用的 Bulk 量是一萬五千個 JSON 文件，其實能更多點以求取更好效率，但是考慮到大規模攻擊時 Netflow 也會爆發導致 Bulk 頻率也增加，因此設定比較小的 Bulk 讓 JVM 可以比較快處理並釋放記憶體好包容頻率較高的 Bulk 避免凍結，必須留下應對極端狀況的餘裕，不能平時就吃滿滿。

當 FL2ELK 子程序對 ES Client Node 作 Bulk 匯入將資料傳輸完後並不能立即斷線離開，此時必須等待 ES 端的回應，ES Client Node 將 Bulk 包展開並轉製成一個個 Document 物件後必須決定每一篇 Document 的 Unique ID，然後根據 ID 算出要送給哪個分片(Shard)，再從目前 Cluster 狀態查出該 Shard 現在哪個 Data Node 上，然後送給該 Data Node，直到該 Node 回覆 Document 已被索引或檢查失敗駁回，ES Client Node 才能標記這一篇 Document 為作完(所以作完不代表成功)，等該 Bulk 包內所有 Document 都做完，然後 ES Client Node 才會一口氣回覆每一筆 JSON 文件的匯入結果與 Document ID(成功的才有 ID)給 Bulk 客戶端，亦即 FL2ELK 子程序，子程序在收完回覆結果後，才能結束 Bulk API 連線，刪除所轄的暫存檔並結束自身。

由於 FL2ELK 母程序僅單純收下 UDP 封包存入暫存檔案，系統記憶體若是足夠大，寫入的暫存檔基本上還在 IO 緩衝區內，而且 ASYNC 非同步檔案系統也不需要真正等待硬碟寫完，因此 FL2ELK 的網路佇列消化速度不會慢於佇列堆積，等到累積到設定量時暫存檔會轉由子程序接手，此時即使發生真正的硬碟 IO 也不會造成母程序等待，因此本環節也不會造成掉包，在此需要注意的是，FL2ELK 的母程序不需要像 FL2JSON 那麼多隻，因為 FL2ELK 要累積一定數量才打包，如果分散太多隻，會導致累積成包的時間拖延太久而失去即時的意義，因此 FL2ELK 母程序個數只要足夠應付不掉包即可，這個數量可經過實際測試後決定一個適當數量。

2.4 增加 ES Client Node 數量

經過前面三小節所提的三關分散措施，每秒十萬筆 Netflow 大約分散成每秒五到十個 Bulk 包，而 ES Cluster 處理每個 Bulk 請求所需時間視當下全系統負載而有很大的變化，當 Cluster 負載很低運行順暢的時候，一個 Bulk 請求可能一兩秒就能完成並回覆，但是大部分的時候並不會這麼順暢，我們實際的經驗在整個系統最順暢時也隨時有數十個 Bulk 在同時處理中，不順時常會堆積成數百上千個。

當有某些 ES Data Node 因為記憶體周轉不靈發生 GC 凍結，就會導致所有正在處理 Bulk 的 ES Client Node 必須等待那個凍結 Node 的回覆而佔用記憶體無法釋放，而新的 Bulk 請求又不斷進來，很快就會形成堆積，而當凍結的 Node 恢復後，堆積的眾多 Bulk 請求會一擁而上，很容易又會造成一些 Data Node 形成 GC 凍結，而 Bulk 卡多了也會不斷消耗 Client Node 記憶體，Client Node 可能被

拖垮，然後 FL2ELK 因為等待 ES 回應的子程序不斷堆積，最終也會被拖垮，這樣的連鎖反應其實蠻容易發生，究其根本原因就是 ES Data Node 不夠多，然而增加 ES Data Node 的成本甚高，因此我們採用增加 ES Client Node(只需要記憶體)數量以及增加 FL2ELK 主機記憶體的方式拉長連鎖反應的腹地縱深，降低全面崩潰的機率，依照我們的經驗，發生連鎖反應時往往會讓 FL2ELK 堆積到兩三千隻等待 Bulk 回覆的子程序，通常暴增到幾千個之後會到達一個頂點，而後隨著 ES Data Node 慢慢緩過氣來，堆積也會逐漸緩解消化，利用上述多種措施，我們實現了將每秒十萬筆等級的 Netflow 大資料流即時匯入 ES 平台中，提供給網路異常使用即時偵測系統可以快速查詢的服務。

當然，如果發生堆積時又有幾個 Data Node 再度發生凍結，那麼瀕臨極限的堆積會很快突破到無法挽回的地步，此時 ES Data Node 就會發生大量凍結與離線，Cluster 就會崩潰，接下來本文將會提出我們對於 ES Cluster 的維運與異常狀態處理方法。

3. ES 平台監控與自動異常處置

一般來說，ES Cluster 是相當穩定的，並不需要花多少心力去維運，但是我們的 ES 平台因為需要持續接收平均每秒五萬筆最大破十萬等級的 Netflow 數據流，並且服務骨幹即時異常使用偵測系統的頻繁查詢，壓力相當大，每天都要新增4到5TB 的資料總量進硬碟，並移除最早的一天資料以騰出空間，線上索引樹的合併抄寫也會一直操勞硬碟，因此系統硬體的勞損遠比一般伺服器更高，更早進入中老年的不穩定期，常遇到硬碟壞軌 IO 失敗造成 JVM 周轉不靈，就像車流很大的公路，只要有台車突然慢停一下，就會造成連鎖反應往後塞住，並且連累旁邊車道也放慢下來，此時若有其他車也出事，就會雪上加霜。此外，網路上常會發生大規模掃描，偶爾會有 DDoS 攻擊，這兩種惡意行為都會造成短時內 Netflow 數量暴增，就像公路突然擠入爆量車流，結果就是塞車回堵，影響許多交流道如骨牌效應般回堵到平面道路，此時就會引發全系統的崩潰。

除了偶發大規模惡意行為帶來的風險，系統定期備份也同樣會帶來風險，按照 ISO 與資安要求，資料必須有備份，所以我們會在每天午夜換日時進行已完檔資料快照備份，因為資料量大，備份傳輸時也是系統的脆弱時間(會擠佔 JVM 記憶體)，而夜間也是許多惡意者喜歡的攻擊時段，因為大部分管理者休息而無法快速反應處理，我們的 ES 平台也偶爾會因為半夜的大規模攻擊 Netflow 暴增遇上備份快照的壓力而導致系統崩潰，因半夜無人處理，使得崩潰的系統一直無法自行修復，因此我們需要開發可以自動監控 ES 平台狀態並自行判斷處置的程式來協助維運，對此我們作了多次嘗試，找到足夠簡單且不容易出意外的操作步驟好讓程式自動執行，將說明如下文。

3.1 ES Cluster 狀態意義與判別

自動監控最主要就是要探查 ES 狀態，RESTful API 如下文所示：

```
RESTful API:
#curl -XGET 'http://es:9200/_cluster/health?pretty'
Respone:
{
  "cluster_name" : "es",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 178,
  "number_of_data_nodes" : 156,
  "active_primary_shards" : 15670,
  "active_shards" : 31340,
  "relocating_shards" : 52,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

如上所示，[status]有 green/yellow/red 三種狀態， red 狀態表示有 primary_shard 找不到，yellow 狀態表示所有 primary shard 都有，但是某些 replication shard 還沒準備好，我們必須明確了解狀態的意義才能正確處置，下一段將仔細說明。

我們因為 Netflow 資料太龐大，因此只作一份複製，舉例一個 Index 拆分成150分片(primary shard)，因此每個 primary shard 只複製一份分片(replication shard)，ES 的 Master Node 會負責指定誰是 primary shard，誰是 replication shard，並且安排哪個 shard 該放置在哪個 Data Node 上，primary 跟 replication 不會放在同一台機器上，replication shard 的資料必須跟 primary shard 看齊。

當某台機器或 Data Node 因故離線時(通常是因為 JVM GC 凍結太久)，Master Node 發現有 primary shard 消失，就會下令把尚存的 replication shard 扶正，同時安排某個 Data Node 產生新的 replication shard 並從剛扶正的 primary shard 複製資料過去，如果消失的是 replication shard，Master Node 也會安排某個 Data Node 產生新的 replication shard 並從 primary shard 複製資料過去，因此，只要發生 Data Node 離線，一分鐘內就會引發許多 Data Node 之間的 Shard 資料複製傳輸，被牽連到的 Data Node 們會因此負載暴增，並擠壓原本的工作，就等於公路開始變慢，塞車風險提高。

如果離線的 Data Node 只有一個，那麼只會造

成 yellow 狀態，因為必然有 replication shard 在其他 Data Node 可以迅速扶正為 primary，但若是多數 Data Node 都離線時，就有可能發生 primary & replication shard 同時消失的不幸狀況，資料無處可補，這就是 red 狀態了，這狀態只能等 primary 或 replication shard 所在的 Data Node 回歸隊列才能解決，若無法回歸資料，則這個 Index 就毀了，red 狀態就不會消失，所以 red 相對 yellow 而言並非是指出事的 shard 數量多寡，而是指資料是否能復原的意思。而 red 狀態因為代表有某塊資料完全空缺，因此查詢與新資料匯入都會被凍結，系統處於無法服務的崩潰狀態。而 yellow 則因為所有的 primary shard 都還在，資料並無缺損，所以各種服務都能提供，只是效能較差且崩潰風險較 green 高。

3.2 自動解除 red 狀態的方法

如果狀態異常時骨幹 Netflow 新增壓力不大，通常會自己慢慢恢復，即時 red 也能自行恢復，因為 Data Node 通常只是因為 JVM 記憶體周轉不過來凍結一段時間，解凍後會自行回歸。然而 Netflow 爆量時會讓所有 Node 都處於緊繃狀態，一個 Data Node 摔倒往往會接連絆倒其他夥伴，倒地的 Data Node 多了就容易發生 red 狀態，因為 red 不能服務，無法收下新匯入的資料，連鎖反應造成 FL2ELK 那邊也塞車堆積，等所有 Data Node 回歸狀態轉 yellow 重新接收匯入時，FL2ELK 那邊堆積等待的龐大資料流會更加猛暴的衝入，結果會瞬間沖垮更多 Data Node，於是又成 red 狀態了，如此陷入惡性循環，使得崩潰的 Data Node 越來越多，系統離自動修復完成越來越遠。

我們觀察後發現半夜的 red 狀態通常很難自動修復，因為正常半夜流量低，會發生 red 表示遇到猛爆式大規模惡意行為，上述連鎖反應的風險很高，而且半夜也無法人工處理，因此我們經過實驗後決定採用「長痛不如短痛」的自動處置，方法如下文。

3.2.1 自動監控程式每五分鐘探查一次 Cluster 狀態(這週期可以更短)，若發現 red 狀態，立即用 RESTful API 停止 replication shard 複製行為，指令如下：

```
# curl -XPUT "es:9200/_cluster/settings?mas-
ter_timeout=5m&pretty" -d
'{
        "persistent": {
                "cluster.routing.allocation.enable":
"primaries"
        },
        "transient": {
                "cluster.routing.allocation.enable":
"primaries"
        }
}'
```

先禁止 replication shard 的複製主要是為了降低全系統的負載，避免目前還存活的 Data Node 一起被拖垮，也避免情勢被搞得更加混亂。

同時，必須停止所有 FL2ELK 的程序不再產生新的 Bulk 請求，這是最主要的降載手段，但此時已經傳給 ES Client Node 的 Bulk 請求仍在等待 Data Node 執行中，大部分在十分鐘或二十分鐘內系統會自動從 red 修復到 yellow 狀態，若系統自動修復到 yellow，則跳到3.2.4小節的情況。

3.2.2 監控程式會追蹤 red 的持續時間，據我們的經驗，如果完全降載後三十分鐘還無法脫離 red 就已經沒有自動修復的希望，表示崩潰的 Data Node 已經形成互相循環拖累的混亂狀況，此時只能重啟全部的 Data Node 以終結混亂。

3.2.3 重啟步驟首先必須先送 SIGNAL TERM 結束所有派發 Netflow Bulk 的 ES Client Node 使堆積資料流消失，否則等 Data Node 重啟完，堆積資料流依然會猛暴衝垮尚未完全修復的 Cluster。接著用 SIGNAL TERM 結束所有 Data Node，然後重啟 Data Node，每個 Data Node 啟動後會開始自檢本身擁有哪些 shards 並向 Master Node 報告，等 Data Node 全數到齊後，Master Node 會參考重啟之前的最後狀態紀錄開始盤點所有 Data Node 的 shard，確認哪些 shard 應該是 primary shard，確認完成後如果所有 primary shard 都存在，全系統狀態就會從 red 恢復到 yellow，此時所有 replication shard 都標記為「unassigned_shards」，其數量會與「active_primary_shards」相等，且此時「active_shards」數量會等於「active_primary_shards」。

3.2.4 監控程式發現系統恢復到 yellow，就會歸零 red 狀態持續時間的紀錄，啟動先前停止的 ES Client Nodes，並下令恢復 replication shard 可複製遷移，如果不下此命令，那麼系統就會一直停在 yellow 狀態無法繼續恢復。指令如下，之後就能見到「unassigned_shards」數量逐漸減少，「active_shards」開始增加。

```
# curl -XPUT "es:9200/_cluster/settings?mas-
ter_timeout=5m&pretty" -d
'{
        "persistent": {
                "cluster.routing.allocation.enable":
null
        },
        "transient": {
                "cluster.routing.allocation.enable":
null
        }
}'
```

3.2.5 當監控程式發現 yellow 狀態下的「unassigned_shards」數量降低到安全範圍，監控程式就會下令 FL2ELK 重新執行 Bulk 匯入，至此，整個自動處置程序完成。

4. 結論

這些年來，TWAREN 頻寬逐代提升，網路流量自然也是越來越高，Netflow 資料流也是越來越大，本文利用一個分發程式接收一個路由器 Netflow 作為第一階分散，每個分發程式轉發給1個或數個 FL2JSON 程式作為第二階分散將 Netflow 轉換為 JSON Document，此階段最消耗 CPU 運算，然後 FL2JSON 再將 JSON Document 轉發給第三關 FL2ELK 程式作堆積收集，然後 FL2ELK 才打包送給 ES Client Node，以上多層次分散處理的首要目標是為了使消化網路輸入佇列的速度能快過新資料進入網路輸入佇列堆積的速度，盡力做到不掉包，目前整個流程我們只動用到三台中階伺服器就足以即時處理每秒十幾萬筆的 Netflow 資料流，顯見我們開發的程式有達到需求目標。

然而尷尬的是，隨著前端處理效能的改善，能處理更大的網路大規模異常行為暴衝資料流通，使得更高的壓力轉移到後端的 ES 大資料平台，導致現有的運轉負載越來越高，常常處於走鋼索的緊繃狀態，就像電器失去了突波保護裝置一樣，ES 大資料平台的穩定度下降，限於資金與成本考量，我們只能暫時以快速自動反應處置的手段降低平台崩潰的風險，我們開發 ES 自動監控程式視異常嚴重狀況分三階「暫停 replication shard 複製搬移」、「暫停 Netflow 資料匯入」、「Data & Client Nodes 重啟」自動處置，使 ES 平台能盡快恢復服務，這些經驗希望能對有類似境況的管理者有所幫助。

參考文獻

[1] https://www.elastic.co/

[2] https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html#wp1006108

[3] https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.pdf

[4] https://www.iana.org/assignments/ipfix/ipfix.xhtml

[5] https://github.com/sleinen/samplicator