add Retrieval system .docx

Service Discovery(服務發現，SD)

以下針對Service Discovery的程式做解析：

若監控的目標時常改變，則每次都需修改靜態配置後再進行reload，為了避免這種情況，Prometheus開發了動態發現(Service Discovery, SD)，能感知目標的CRUD後自動reload，而不需重新配置再重啟server。與SD相關的資訊傳遞至discovery manager，然後discovery manager會和SD系統進行通訊，當使用者下達reload handler的指令時，會調用reloadConfig ，並依次調用相關function [[[1]](#footnote-1)][[[2]](#footnote-2)][[[3]](#footnote-3)] (圖 1)。但當SD的配置進行改動時(例如新增job)，prometheus還是需要手動reload配置文件並重啟server。

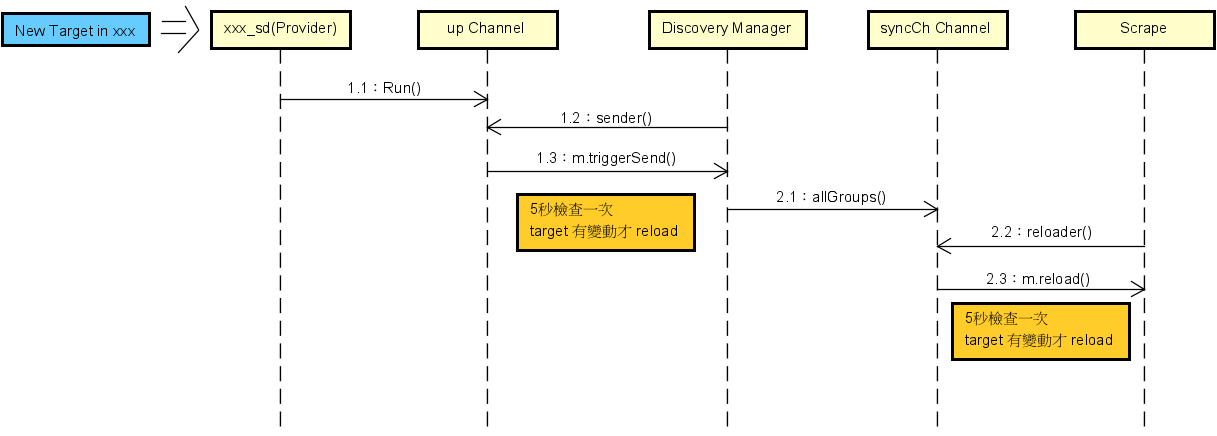


圖 1：Service Discovery Sequence Diagram

SD機制的目的是發現provider(e.g. DNS, Azure, Kubernetes)的target並最終將資訊提供給Prometheus監控。SD的Discoverer介面(圖 2)提供給各Provider介接 [[[4]](#footnote-4)]，最開始會調用Run()(圖 3)將監控的target group發送至up channel，再用init()的discovery.RegisterConfig(圖 4)進行註冊，接著可能將整組或是有更動(視provider而定)的target group透過up chan發送給discovery Manager處理。

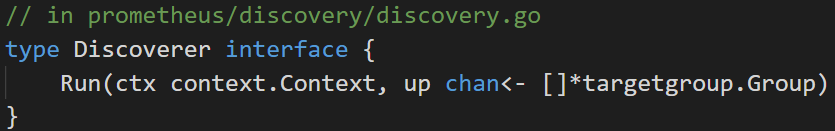


圖 2：Discoverer Interface

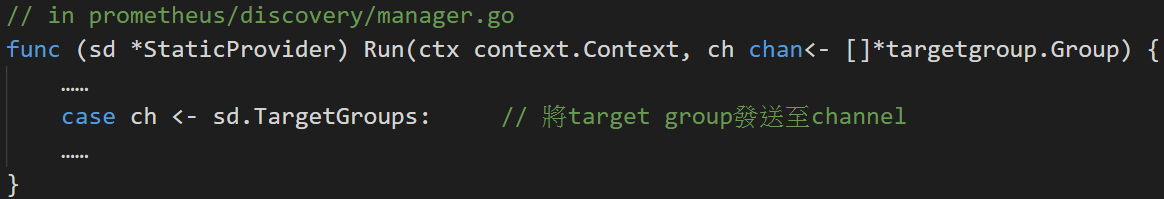


圖 3：Run()本體

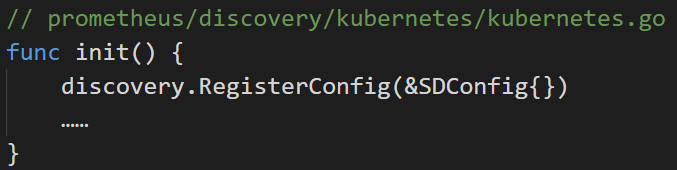


圖 4：init()內的discovery.RegisterConfig

Group(圖 5)是一群target的list，其擁有共同的LabelSet，LabelSet是一組名稱與值的map [[[5]](#footnote-5)]

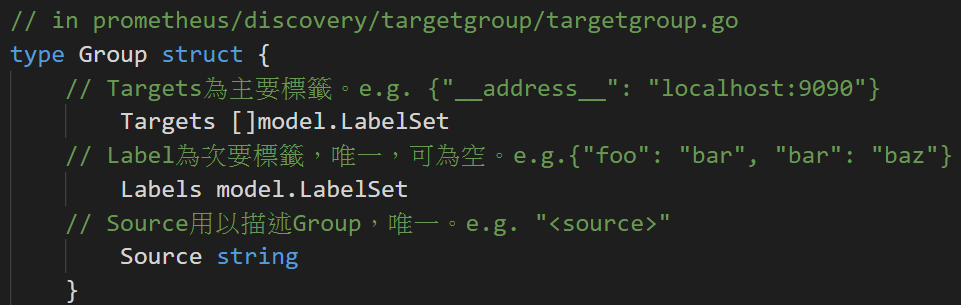


圖 5：Group struct

Discovery Manager

Manager(圖 6)會處理由up Chan送來的資料(監聽與獲取資料方式與scrape處理sync Chan雷同，故此略)，並透過poolKey(對應配置文件的job name)和provider快速找到對應的target group陣列。

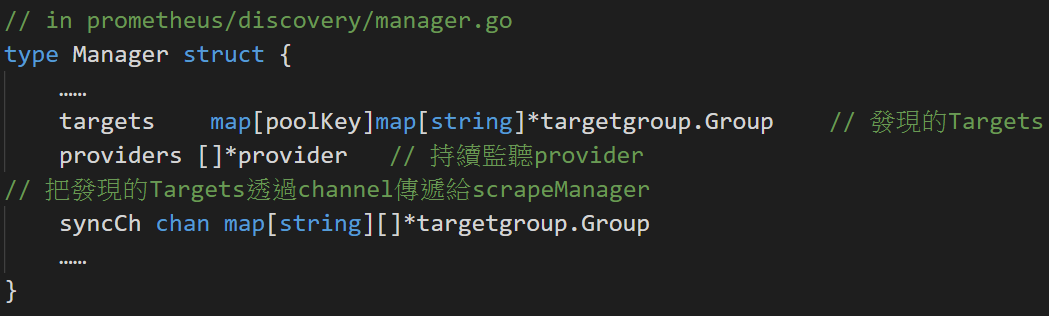


圖 6：Manager

如此在discovery manager裡就可以拿到所有targets的資訊，接著discovery manager透過allGroups() [[[6]](#footnote-6)]傳入syncCh，scrape manager會持續監聽syncCh，一旦有新message傳入，scrape就會reload變動的target group(圖 7)，之後scrape就會開始pull target進入TSDB。



圖 7：scrape監聽syncCh

每5s監聽m.triggerReload信號，執行m.reload() [[[7]](#footnote-7)]加載targets(圖 8)。

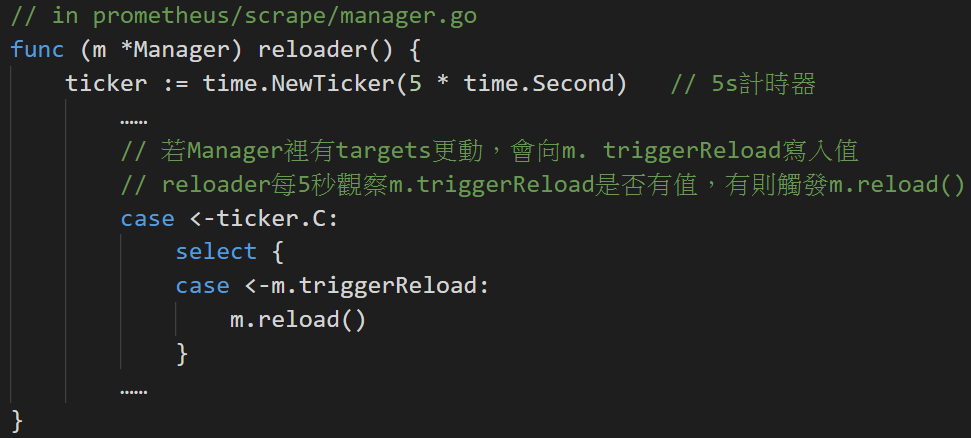


圖 8：Manager

TSDB

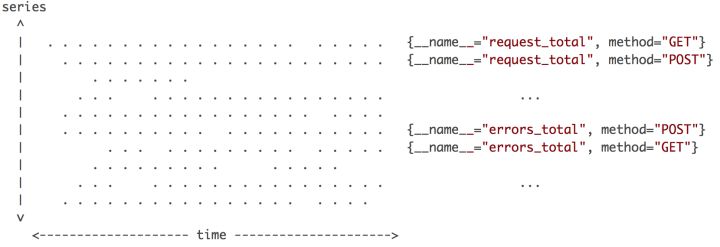
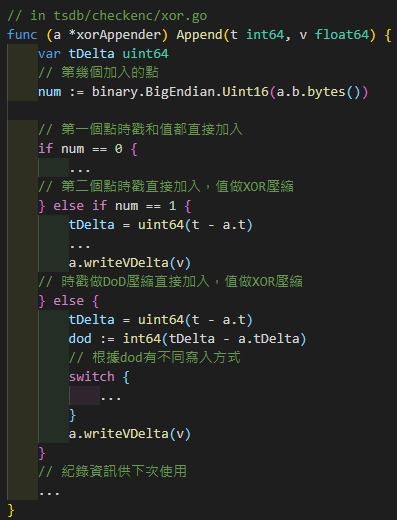


圖 9：series 紀錄

在tsdb中，同一時間會有多個series監控紀錄指定資源，一個series由數據點的list和label的list組成。數據點由一個時戳和一個值組成，而label為一key-value pair。將一個新的數據點加入series，而在加入時會先將時戳和值分別用DoD壓縮和XOR壓縮再加到series內(圖 10)。因為tsdb有以下特性，一、 相鄰數據點的時戳差距變化， 即使有浮動也僅在小範圍內(採樣間隔固定)；二、 相鄰數據點的value變化也很小，甚至有相當比例為0。因此採用此種壓縮能有效壓縮series的大小。

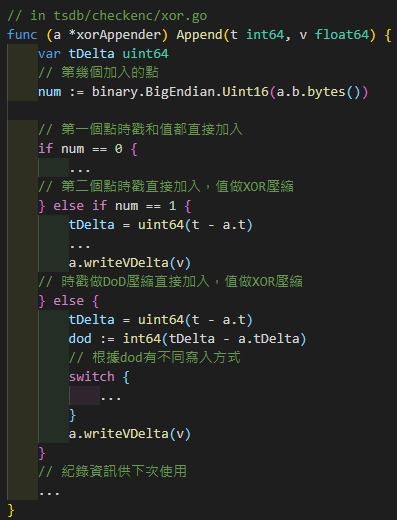


圖 10：數據點加入series

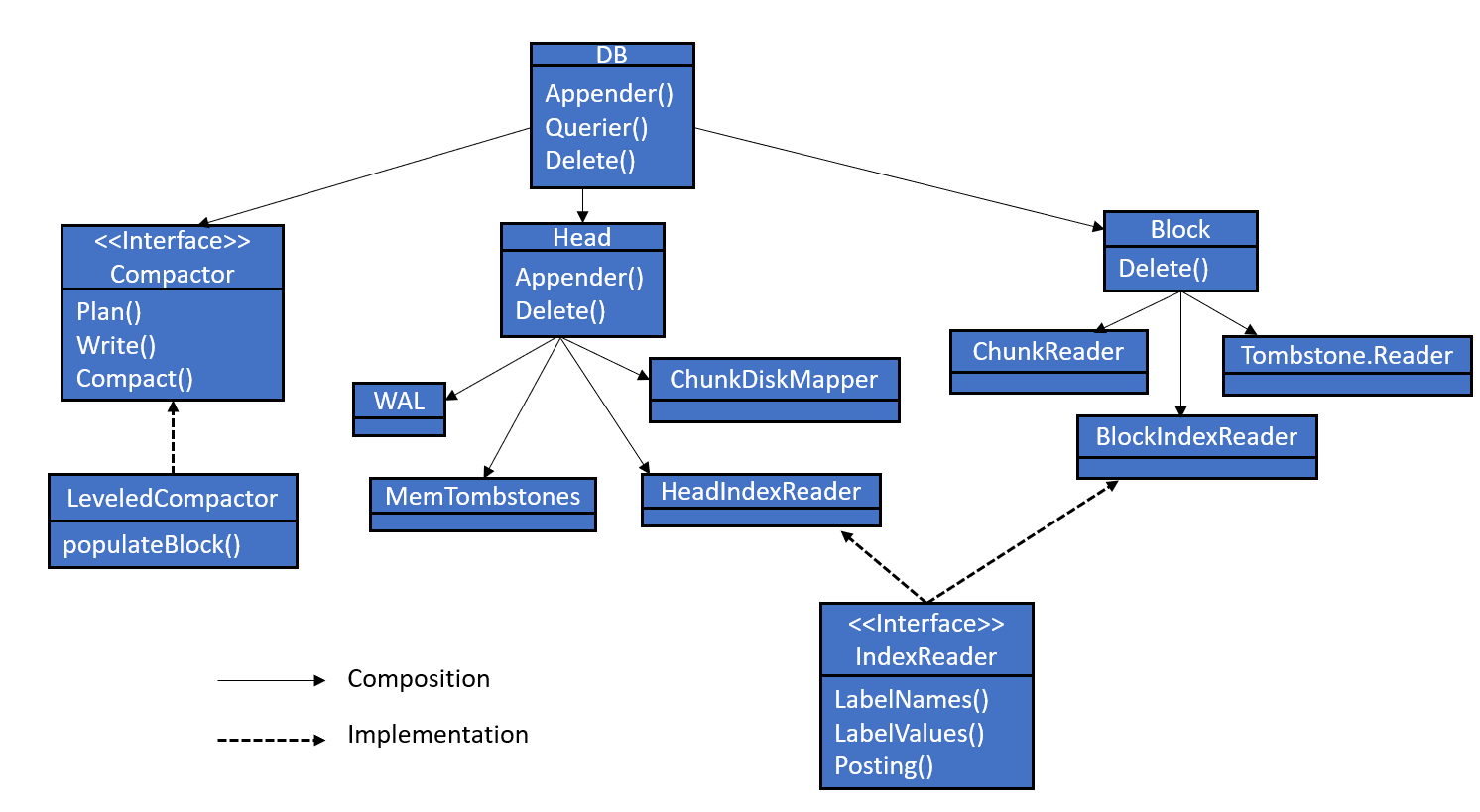


圖 11：TSDB架構

整個DB(圖 11)採用類似LSM algorithm(Log Structured Merge-tree)，主要由Head, Block, Compactor(圖 12)物件組成。Head指的是正在寫入的block，儲存在memory中，且因為在tsdb中有「越近期資料，越容易被查找」的特性，因此僅將Head存進memory既能減少空間也能加快查訪速度；Block是已經持久化且無法更動(immutable)的block。不管是head或是block，都以更小的chunk為單位保存在disk。

建立資料時，為避免資料因為意外崩潰導致數據丟失，除了經過前述壓縮後寫進head block，也會將未壓縮的資料寫入至預寫式日誌(Write-ahead Logging)中。當寫入head block超過寫入時間時，會將其持久化成一個level 0 block；當同一level的block夠多，Compactor就會壓縮成level更高的block。

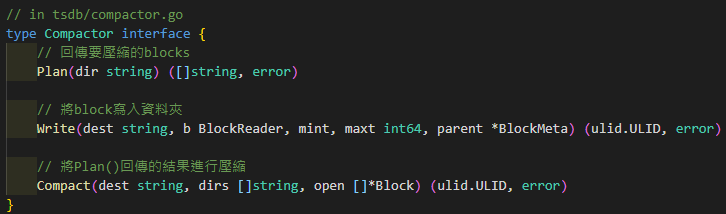


圖 12：Compactor Interface

讀取時，BlockQuerier根據不同結構產生headIndexReader或是blockIndexReader(圖 13)，index是倒序索引(inverted index)的資料結構，將給定的label name映射到目標所在chunk的offset，減少查找所需時間。

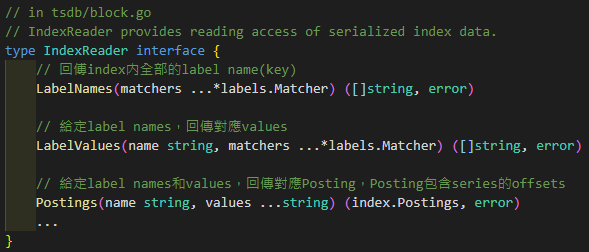


圖 13：IndexReader Interface

更新資料時，若目標在block中，因為block無法做更動，所以只能寫在head，在讀取時會以新的為主，而壓縮block時若有對同一資料進行寫入，會只有新的資料寫入至壓縮後的block。

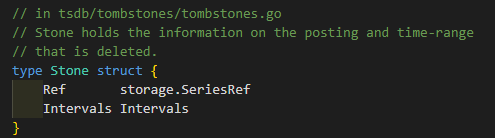
刪除資料時，刪除紀錄會保存在目標所擁有的tombstone files(圖 14)，而非立即從block刪除。當整個block的資料都超過保留時間後，整個block就會被丟棄。

圖 14：Stone struct

1. [] <https://github.com/prometheus/prometheus/blob/063319087c122b3b296cc630d93f577dac31fd1c/cmd/prometheus/main.go#L874-L909> [↑](#footnote-ref-1)
2. [] <https://github.com/prometheus/prometheus/blob/063319087c122b3b296cc630d93f577dac31fd1c/cmd/prometheus/main.go#L1159-L1201> [↑](#footnote-ref-2)
3. [] <https://github.com/prometheus/prometheus/blob/063319087c122b3b296cc630d93f577dac31fd1c/cmd/prometheus/main.go#L662-L748> [↑](#footnote-ref-3)
4. [] <https://github.com/prometheus/prometheus/tree/main/discovery> [↑](#footnote-ref-4)
5. [] <https://github.com/prometheus/common/blob/840c039c5fcce8204ed656bd75b084d2e9d80c1d/model/labelset.go#L28> [↑](#footnote-ref-5)
6. [] <https://github.com/prometheus/prometheus/blob/6555cc68caf8d8f323056e497ae7bb1e32a81667/discovery/manager.go#L381-L399> [↑](#footnote-ref-6)
7. [] <https://github.com/prometheus/prometheus/blob/063319087c122b3b296cc630d93f577dac31fd1c/scrape/manager.go#L188-L216> [↑](#footnote-ref-7)