技術顏良

隨筆- 2554 文章- 0 評論- 67 閱讀- 314萬

使用vmagent 取代Prometheus 採集監控指標

vmagent 可以幫助我們從各種來源收集指標並將它們儲存在 VM 或任何其他支援remote write 協定的 Prometheus 相容的儲存系統中。

特性

vmagent 相較於Prometheus 抓取指標來說具有更多的靈活性，例如除了拉取（pull）指標還可以推送（push）指標，此外還有很多其他特性：

- 可以替換prometheus 的scraping target
- 支援從Kafka 讀寫數據
- 支援基於prometheus relabeling 的模式新增、移除、修改labels，可以在資料傳送到遠端儲存之前進行資料的過濾
- 支援多種資料協議，influx line 協議，graphite 文字協議，opentsdb 協議，prometheus remote write 協議，json lines 協議，csv 資料等
- 支援收集資料的同時，並複製到多種遠端儲存系統
- 支援不可靠遠端存儲，如果遠端存儲不可用，收集的指標會在 `-remoteWrite.tmpDataPath` 緩衝，一旦與遠端存儲的連接被修復，緩衝的指標就會被發送到遠端存儲，緩衝區的最大磁碟用量可以 `-remoteWrite.maxDiskUsagePerURL` 用來限制。
- 相比prometheus 使用更少的記憶體、cpu、磁碟io 以及網路頻寬
- 當需要抓取大量目標時，抓取目標可以分散到多個vmagent 實例中
- 可以透過在抓取時間和將其發送到遠端儲存系統之前限制唯一時間序列的數量來處理高基數和高流失率問題
- 可以從多個檔案載入scrape 配置

公告

暱稱： 技術顏良
園齡： 7年3個月
粉絲： 212
追蹤： 26
+加關注

< 2025年3月 >

日	一	二	三	四	五	六
23	24	25	26	27	28	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

搜尋

找找看

常用連結

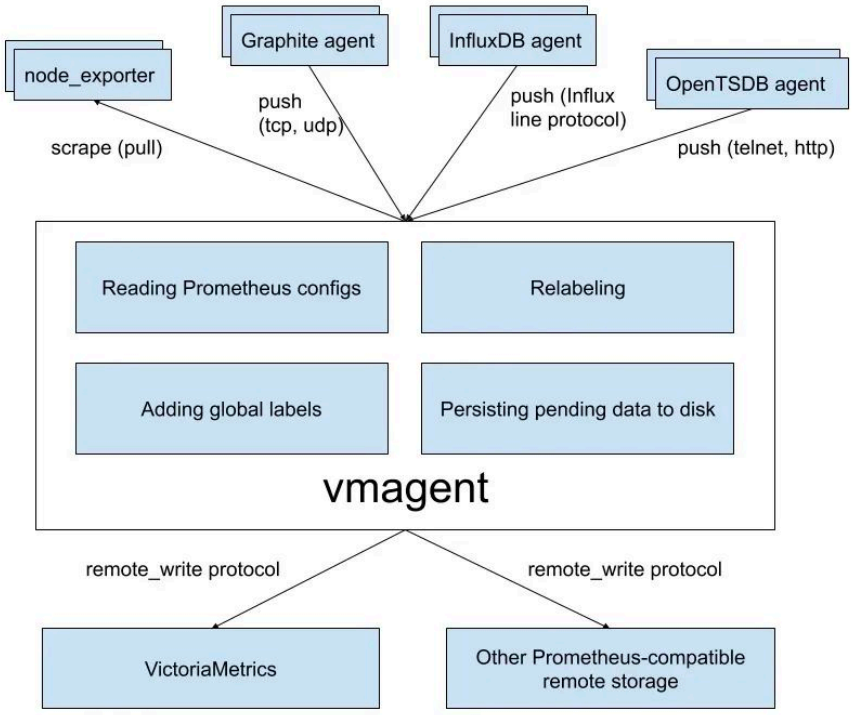
我的隨筆
我的評論
我的參與
最新評論
我的標籤

我的標籤

zabbix (1)
ELK (1)
Automa (1)

隨筆分類

ansible(51)
ansible/awx(4)
antd-mobile(1)
argocd(2)
bash(5)
Caddy(1)
centos6(1)
Ceph(12)
chrony(4)
Cloudera Enterprise 6.2.0(1)
confluence (1)
Consul(4)



部署

接下來我們以抓取Kubernetes 叢集指標為例說明如何使用vmagent，我們在這裡使用自動發現的方式來進行設定。 vmagent 是相容prometheus 中的 `kubernetes_sd_configs` 配置的，所以我們同樣可以使用。

要讓vmagent 自動發現監控的資源對象，需要存取APIServer 取得資源對象，所以首先需要設定rbac權限，建立如下所示的資源清單。

```
# vmagent-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vmagent
  namespace: kube-vm
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: vmagent
rules:
  - apiGroups: ["", "networking.k8s.io", "extensions"]
    resources:
      - nodes
      - nodes/metrics
      - services
      - endpoints
      - endpointslices
      - pods
      - app
      - ingresses
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources:
      - namespaces
      - configmaps
    verbs: ["get"]
  - nonResourceURLs: ["/metrics", "/metrics/resources"]
    verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: vmagent
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

corosync+pacemaker 高可用(5)
DDOS(1)
DevOps(5)
更多

隨筆檔案

2025年2月(4)
2025年1月(5)
2024年12月(30)
2024年11月(16)
2024年10月(37)
2024年9月(50)
2024年8月(41)
2024年7月(66)
2024年6月(61)
2024年5月(25)
2024年4月(46)
2024年3月(34)
2024年2月(34)
2024年1月(68)
2023年12月(36)
更多

閱讀排行榜

1. 快速搭建ELK日誌分析系統(138244)
2. 設定Redis最大佔用記憶體(81234)
3. k8s 超詳細總結(79660)
4. Nginx 效能最佳化有這篇就夠了！ (65837)
5. 常用nginx rewrite重定向-跳轉實例:(55815)

評論排行榜

1. mysql8.0設定忽略大小寫後無法啟動(3)
2. 理解Golang元件protobuf (2)
3. k8s雲端原生分散式區塊儲存--Longhorn 初步體驗(2)
4. k8s 超詳細總結(2)
5. 快速搭建ELK日誌分析系統(2)

推薦排行榜

1. Nginx 效能最佳化有這篇就夠了!(8)
2. 快速搭建ELK日誌分析系統(6)
3. 設定Redis最大佔用記憶體(5)
4. k8s 超詳細總結(4)
5. python中的cls到底指的是什麼(4)

最新評論

1. Re:10個最佳開源智慧家庭系統（SHS)
大哥增加一些比較啊，
--一點不懂
2. Re:手把手搭建自己私有的MQTT伺服器，完成裝置上雲
圖片消失術啊
--zyinnnnn
3. Re:k8s一條預設配置導致cpu升高
大佬評測下
github點com/minchieh-fay/har

```
  name: vmagent
subjects:
  - kind: ServiceAccount
    name: vmagent
    namespace: kube-vm
```

然後加入vmagent 配置，我們先只配置自動發現Kubernetes 節點的任務，建立如下所示的ConfigMap物件：

```
# vmagent-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: vmagent-config
  namespace: kube-vm
data:
  scrape.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s

    scrape_configs:
    - job_name: nodes
      kubernetes_sd_configs:
        - role: node
      relabel_configs:
      - source_labels: [__address__]
        regex: "(.*):10250"
        replacement: "${1}:9111"
        target_label: __address__
        action: replace
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
```

這裡我們透過自動發現Kubernetes 節點來取得節點監控指標，需要注意 `node` 這種role 的自動發現預設取得的是節點的 `10250` 端口，這裡我們需要透過 `relabel` 將其 `replace` 為 `9111` 。

然後新增vmagent 部署資源清單，如下所示：

```
# vmagent-deploy.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vmagent-pvc
  namespace: kube-vm
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs-client
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmagent
  namespace: kube-vm
  labels:
    app: vmagent
spec:
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
      labels:
        app: vmagent
    spec:
      serviceAccountName: vmagent
      containers:
        - name: agent
          image: "victoriametrics/vmagent:v1.77.0"
          imagePullPolicy: IfNotPresent
          args:
            - -promscrape.config=/config/scrape.yml
            - -remoteWrite.tmpDataPath=/tmpData
            - -remoteWrite.url=http://vminsert:8480/insert/0/prometheus
```

這個很好用, 還不需要3個master做容災, 2個節點也能容災

--dota2愛好者

4. Re:詳解Go 中的rune 類型

部落客最近也在學習go，小白一個，在看godotenv 專案中，解析自訂設定檔時候用到rune， 我看大體思路是讀取到文字文件，拿到[]byte 切片數據， 然後取得每行的切片數據，然後再根據...

--望天hous

5. Re:k8s二進位方式部署

文章裡的圖片都不顯示哦

--雪、けがをする

```
            - -envflag.enable=true
            - -envflag.prefix=VM_
            - -loggerFormat=json
          ports:
            - name: http
              containerPort: 8429
          volumeMounts:
            - name: tmpdata
              mountPath: /tmpData
            - name: config
              mountPath: /config
      volumes:
        - name: tmpdata
          persistentVolumeClaim:
            claimName: vmagent-pvc
        - name: config
          configMap:
            name: vmagent-config
```

我們將vmagent 配置透過ConfigMap 掛載到容器 `/config/scrape.yml` ，另外透過 `-remoteWrite.url=http://vminsert:8480/insert/0/prometheus` 指定遠端寫入的位址，這裡我們寫入前面的vminsert 服務，另外有一個參數 `-remoteWrite.tmpDataPath` ，該路徑會在遠端儲存不可用的時候用來快取收集的指標，當遠端儲存修復後，快取的指標就會被正常傳送到遠端寫入，所以最好持久化該目錄。

集群模式

單一vmagent 實例可以抓取數萬個抓取目標，但是有時由於CPU、網路、記憶體等方面的限制，這還不夠。在這種情況下，抓取目標可以在多個vmagent 實例之間進行拆分。叢集中的每個vmagent 實例必須使用具有不同 `-promscrape.cluster.memberNum` 值的相同 `-promscrape.config` 配置文件，該參數值必須在 `0 ... N-1` 範圍內，其中 `N` 是叢集中vmagent 實例的數量。集群中vmagent 實例的數量必須傳遞給 `-promscrape.cluster.membersCount` 命令列標誌。例如，下列命令可以在兩個vmagent 實例的叢集中傳播抓取目標：

```
vmagent -promscrape.cluster.membersCount=2 -promscrape.cluster.memberNum=0 -promscrape.
vmagent -promscrape.cluster.membersCount=2 -promscrape.cluster.memberNum=1 -promscrape.
```

當vmagent 在Kubernetes 中執行時，可以 `-promscrape.cluster.memberNum` 設定為StatefulSet pod 名稱，pod 名稱必須以 `0 ... promscrape.cluster.memberNum-1` 範圍內的數字結尾，例如， `-promscrape.cluster.memberNum=vmagent-0` 。

預設情況下，每個抓取目標僅由叢集中的單一vmagent 實例抓取。如果需要在多個vmagent 實例之間複製抓取目標，則可以透過 `-promscrape.cluster.replicationFactor` 參數設定為所需的副本數。例如，下列命令啟動一個包含三個vmagent 實例的集群，其中每個目標由兩個vmagent 實例抓取：

```
vmagent -promscrape.cluster.membersCount=3 -promscrape.cluster.replicationFactor=2 -pro
vmagent -promscrape.cluster.membersCount=3 -promscrape.cluster.replicationFactor=2 -pro
vmagent -promscrape.cluster.membersCount=3 -promscrape.cluster.replicationFactor=2 -pro
```

需要注意的是如果每個目標被多個vmagent 實例抓取，則必須在 `-remoteWrite.url` 指向的遠端儲存上啟用重複資料刪除。

所以如果你抓取的監控目標非常大，那麼我們建議使用vmagent 叢集模式，那麼可以使用StatefulSet 方式進行部署

```
# vmagent-sts.yaml
apiVersion: v1
kind: Service
metadata:
  name: vmagent
  namespace: kube-vm
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "8429"
spec:
  selector:
    app: vmagent
  clusterIP: None
  ports:
    - name: http
      port: 8429
      targetPort: http
---
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vmagent
  namespace: kube-vm
  labels:
    app: vmagent
spec:
  replicas: 2
  serviceName: vmagent
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
      labels:
        app: vmagent
    spec:
      serviceAccountName: vmagent
      containers:
        - name: agent
          image: victoriametrics/vmagent:v1.77.0
          imagePullPolicy: IfNotPresent
          args:
            - -promscrape.config=/config/scrape.yml
            - -remoteWrite.tmpDataPath=/tmpData
            - -promscrape.cluster.membersCount=2
            # - -promscrape.cluster.replicationFactor=2 # 可以配置副本数
            - -promscrape.cluster.memberNum=$(POD_NAME)
            - -remoteWrite.url=http://vminsert:8480/insert/0/prometheus
            - -envflag.enable=true
            - -envflag.prefix=VM_
            - -loggerFormat=json
          ports:
            - name: http
              containerPort: 8429
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
          volumeMounts:
            - name: tmpdata
              mountPath: /tmpData
            - name: config
              mountPath: /config
      volumes:
        - name: config
          configMap:
            name: vmagent-config
  volumeClaimTemplates:
    - metadata:
        name: tmpdata
      spec:
        accessModes:
          - ReadWriteOnce
        storageClassName: nfs-client
        resources:
          requests:
            storage: 1Gi
```

我們在這裡就使用StatefulSet 的形式來管理vmagent，直接應用上面的資源：

```
# 先将前面示例中的 prometheus 停掉
☸ ➜ kubectl scale deploy prometheus --replicas=0 -n kube-vm
☸ ➜ kubectl apply -f vmagent-rbac.yaml
☸ ➜ kubectl apply -f vmagent-config.yaml
☸ ➜ kubectl apply -f vmagent-sts.yaml
☸ ➜ kubectl get pods -n kube-vm -l app=vmagent
NAME         READY   STATUS    RESTARTS   AGE
vmagent-0    1/1     Running   0          3m43s
vmagent-1    1/1     Running   0          2m9s
```

這裡我們部署了兩個vmagent 實例來抓取監控指標，我們這裡一共3 個節點。

```
☸ ➜ kubectl get nodes
NAME      STATUS   ROLES                  AGE   VERSION
master1   Ready    control-plane,master   44d   v1.22.8
```

```
node1     Ready    <none>                 44d   v1.22.8
node2     Ready    <none>                 44d   v1.22.8
```

所以兩個vmagent 實例會分別採集部分指標，我們可以透過查看日誌來進行驗證：

```
☸ ➜ kubectl logs -f vmagent-0 -n kube-vm
# ......
{"ts":"2022-05-10T04:44:44.004Z","level":"info","caller":"VictoriaMetrics/lib/promscrap
{"ts":"2022-05-10T04:44:44.006Z","level":"info","caller":"VictoriaMetrics/lib/promscrap
☸ ➜ kubectl logs -f vmagent-1 -n kube-vm
# ......
{"ts":"2022-05-10T04:46:17.893Z","level":"info","caller":"VictoriaMetrics/lib/promscrap
```

從日誌可以看出 `vmagent-0` 實例發現了2 個targets， `vmagent-1` 實例發現了1 個targets，這也符合我們預期的。

接下來我們再新增其他內容的監控，像是APIServer、容器等等，設定如下圖：

```
# vmagent-config2.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: vmagent-config
  namespace: kube-vm
data:
  scrape.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s

    scrape_configs:

    - job_name: nodes
      kubernetes_sd_configs:
        - role: node
      relabel_configs:
      - source_labels: [__address__]
        regex: "(.*):10250"
        replacement: "${1}:9111"
        target_label: __address__
        action: replace
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: apiserver
      scheme: https
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
      - action: keep
        regex: default;kubernetes;https
        source_labels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_service_name
        - __meta_kubernetes_endpoint_port_name

    - job_name: cadvisor
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
      - replacement: /metrics/cadvisor
        target_label: __metrics_path__

    - job_name: endpoints
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
```

```
      - action: drop
        regex: true
        source_labels:
        - __meta_kubernetes_pod_container_init
      - action: keep_if_equal
        source_labels:
        - __meta_kubernetes_service_annotation_prometheus_io_port
        - __meta_kubernetes_pod_container_port_number
      - action: keep
        regex: true
        source_labels:
        - __meta_kubernetes_service_annotation_prometheus_io_scrape
      - action: replace
        regex: (https?)
        source_labels:
        - __meta_kubernetes_service_annotation_prometheus_io_scheme
        target_label: __scheme__
      - action: replace
        regex: (.+)
        source_labels:
        - __meta_kubernetes_service_annotation_prometheus_io_path
        target_label: __metrics_path__
      - action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        source_labels:
        - __address__
        - __meta_kubernetes_service_annotation_prometheus_io_port
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+)
      - source_labels:
        - __meta_kubernetes_pod_name
        target_label: pod
      - source_labels:
        - __meta_kubernetes_namespace
        target_label: namespace
      - source_labels:
        - __meta_kubernetes_service_name
        target_label: service
      - replacement: ${1}
        source_labels:
        - __meta_kubernetes_service_name
        target_label: job
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_node_name
        target_label: node
```

大部分的配置在前面Prometheus 章節都介紹過了，核心就是透過來 `relabel_configs` 控制抓取的任務，vmagent 是兼容傳統的prometheus 重新標記規則的，但也有一些獨特的action，比如上面配置中我們使用了一個的 `keep_if_equal` 操作，該操作的意思是如果指定的標籤值相等則將該條數據保留下來。

有時，如果某個指標包含兩個具有相同值的標籤，則需要刪除它。這可以透過vmagent 支援的 `drop_if_equal` 操作來完成。例如，如果下列relabel 規則包含 `real_port` 和 `required_port` 相同的標籤值，則它會刪除指標：

```
- action: drop_if_equal
  source_labels: [real_port, needed_port]
```

此規則將刪除以下指標： `foo{real_port="123",needed_port="123"}` ，但會保留以下指標： `foo{real_port="123",needed_port="456"}` 。

有時可能需要只對指標子集套用relabel，在這種情況下，可以將 `if` 選項新增至 `relabel_configs` 規則中，例如以下規則僅將 `{foo="bar"}` 標籤新增至與 `metric{label=~"x|y"}` 序列選擇器相符的指標：

```
- if: 'metric{label=~"x|y"}'
  target_label: "foo"
  replacement: "bar"
```

`if` 選項可以簡化傳統的 `relabel_configs` 規則，例如，以下規則可以刪除與 `foo{bar="baz"}` 序列選擇器相符的指標：

```
- if: 'foo{bar="baz"}'
  action: drop
```

這相當於以下傳統的規則：

```
- action: drop
  source_labels: [__name__, bar]
  regex: "foo;baz"
```

不過要注意的是Prometheus 還不支援 `if` 選項，現在只支援VictoriaMetrics。

現在更新vmagent 的配置。

```
☸ → kubectl apply -f vmagent-config2.yaml
```

配置刷新有兩種方式：

- 發送SUGHUP 訊號給vmagent 進程
- 向 `http://vmagent:8429/-/reload` 發送一個http 請求

刷新後就可以開始採集上面的指標了，同樣我們也可以透過來 `http://vmselect/select/0/vmui/` 存取vmui，例如現在我們來查詢pod 的記憶體使用率，可以使用如下的查詢語句：

```
sum(container_memory_working_set_bytes{image!=""}) by(namespace, pod) / sum(container_s
```

vmagent 作為採集指標重要的一環，當然對它的監控也必不可少。 vmagent 透過 `http://vmagent:8429/metrics` 揭露了許多指標，例如 `vmagent_remotewrite_conns` 遠端儲存連接，

`vm_allowed_memory_bytes` 可使用的記憶體大小，我們把一些重要的指標收集起來，透過Grafana 來展示，能夠更好的幫助我們分析vmagent 的狀態。

我們可以使用https://grafana.com/grafana/dashboards/12683 來展示vmagent 的狀態

分類: k8s維護篇

«上一篇: 查詢資料庫表名，資料表信息，MySQL Key值（PRI, UNI, MUL）的含義
»下一篇: Golang 五種原子性操作的用法詳解

posted @ 2022-05-11 12:29 技術顏良 閱讀(290) 評論(0) 編輯 收藏 舉報