



VictoriaMetrics 部署教程



一文搞懂 VictoriaMetrics 的使用-...
PDF

参考pdf，詳細一步步教我改怎麼上手VictoriaMetrics



要上手使用 **VictoriaMetrics**，以下是一个逐步指导，基于您提供的文档 ：

1. 确定使用目标

VictoriaMetrics 提供两种模式：

- **单节点模式**：适合小型环境，低于 100 万数据点/秒。
- **集群模式**：适合大型环境，支持水平扩展。

首先确认您的监控规模以及是否需要高可用架构。

2. 准备运行环境

软件与工具

- **Kubernetes 集群**（建议用于生产环境）
- Docker 或直接二进制文件（用于快速试用）
- Helm（可选，用于快速部署）

3. 单节点模式部署

如果您的需求简单，单节点模式即可满足：

步骤

1. 创建命名空间

```
kubectl create namespace kube-vm
```

2. 部署单节点 VictoriaMetrics 创建 `vm-single.yaml` 配置文件，内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: victoria-metrics
  namespace: kube-vm
spec:
```





```
selector:
  matchLabels:
    app: victoria-metrics
template:
  metadata:
    labels:
      app: victoria-metrics
  spec:
    volumes:
      - name: storage
        persistentVolumeClaim:
          claimName: victoria-metrics-data
    containers:
      - name: vm
        image: victoriametrics/victoria-metrics:v1.91.3
        args:
          - -storageDataPath=/var/lib/victoria-metrics-data
          - -retentionPeriod=1w
        ports:
          - containerPort: 8428
            name: http
        volumeMounts:
          - mountPath: /var/lib/victoria-metrics-data
            name: storage
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: victoria-metrics-data
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 20Gi
  local:
    path: /data/k8s/vm
---
apiVersion: v1
kind: Service
metadata:
  name: victoria-metrics
  namespace: kube-vm
spec:
  type: NodePort
  ports:
    - port: 8428
  selector:
    app: victoria-metrics
```

3. 应用配置

```
kubectl apply -f vm-single.yaml
```

4. 验证服务 检查服务状态并访问 Web UI :

```
kubectl get svc victoria-metrics -n kube-vm
```



4. 配置远程写入

将 Prometheus 的数据远程写入 VictoriaMetrics。

修改 Prometheus 配置

更新 `remote_write` 参数，示例如下：

```
remote_write:
- url: http://victoria-metrics:8428/api/v1/write
```

重启 Prometheus 或执行热加载：

```
curl -X POST "http://<Prometheus-IP>:9090/-/reload"
```

5. 替换 Prometheus

为了减少资源占用，可以完全用 VictoriaMetrics 替代 Prometheus：

1. 停止 Prometheus

```
kubectl scale deploy prometheus --replicas=0 -n kube-vm
```

2. 将 Prometheus 配置挂载到 VictoriaMetrics 增加 `promscrape.config` 参数，例如：

```
args:
- -promscrape.config=/etc/prometheus/prometheus.yaml
```

3. 更新配置并重启 VictoriaMetrics

```
kubectl apply -f vm-single.yaml
```

6. 使用 Grafana

1. 配置 VictoriaMetrics 为数据源：

- 地址：`http://<VictoriaMetrics-IP>:8428`
- 数据源类型：Prometheus

2. 导入 Dashboard（如 Node Exporter 的 ID 为 `16098`）。

7. 集群模式部署（高级用法）

当需要高可用或水平扩展时，切换到集群模式：



- 包含组件：`vminsert`、`vmselect`、`vmstorage`。
- 部署方式：建议使用 Helm 或 Kubernetes 手动部署。

如果需要更具体的配置文件或调整，请告诉我，我可以进一步帮助优化！



参考pdf，詳細一步步教我改怎麼上手VictoriaMetrics集群模式部署



以下是基于PDF文档  的 **VictoriaMetrics 集群模式** 部署逐步指南。

1. 了解集群模式架构

VictoriaMetrics 集群由以下组件组成：

- **vmstorage**：负责存储和查询数据。
- **vminsert**：接收数据并分发到不同的 `vmstorage` 节点。
- **vmselect**：处理查询请求，从 `vmstorage` 获取数据。
- 其他组件：
 - **vmalert**（可选）：报警管理。
 - **vmagent**（可选）：数据采集。

2. 准备工作

系统要求

- Kubernetes 集群
- 确保有足够的存储资源（`PersistentVolume`）
- 配置 Helm 或直接使用 YAML 部署

工具

- **kubectl**：管理 Kubernetes
- Helm（可选）：简化部署过程

3. 部署 VictoriaMetrics 集群模式

步骤 1：部署 `vmstorage`（存储节点）

使用 `StatefulSet` 部署 `vmstorage` 节点。以下是配置文件模板：





```
# vmstorage.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vmstorage
  namespace: kube-vm
spec:
  serviceName: cluster-vmstorage
  replicas: 2 # 设置为2个副本
  selector:
    matchLabels:
      app: vmstorage
  template:
    metadata:
      labels:
        app: vmstorage
    spec:
      containers:
        - name: vmstorage
          image: victoriametrics/vmstorage:v1.91.3-cluster
          args:
            - "--retentionPeriod=1"
            - "--storageDataPath=/storage"
          ports:
            - name: http
              containerPort: 8482
          volumeMounts:
            - name: storage
              mountPath: /storage
      volumeClaimTemplates:
        - metadata:
            name: storage
          spec:
            accessModes: ["ReadWriteOnce"]
            resources:
              requests:
                storage: 50Gi
```

应用配置：

```
kubectl apply -f vmstorage.yaml
```

步骤 2：部署 `vminsert`（数据接入节点）

配置 `vminsert` 用于接收数据并将其分发到 `vmstorage` 节点。

```
# vminsert.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: kube-vm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
```



```
labels:
  app: vminsert
spec:
  containers:
  - name: vminsert
    image: victoriametrics/vminsert:v1.91.3-cluster
    args:
      - "--storageNode=cluster-vmstorage:8400" # 连接到 vmstorage
    ports:
      - name: http
        containerPort: 8480
```

应用配置：

```
kubectl apply -f vminsert.yaml
```

步骤 3：部署 `vmselect`（查询节点）

配置 `vmselect` 用于处理查询请求。

```
# vmselect.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmselect
  namespace: kube-vm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vmselect
  template:
    metadata:
      labels:
        app: vmselect
    spec:
      containers:
      - name: vmselect
        image: victoriametrics/vmselect:v1.91.3-cluster
        args:
          - "--storageNode=cluster-vmstorage:8401" # 连接到 vmstorage
        ports:
          - name: http
            containerPort: 8481
```

应用配置：

```
kubectl apply -f vmselect.yaml
```

步骤 4：创建服务

为每个组件创建 `Service`，以暴露其接口。

```
# 服务配置示例
apiVersion: v1
kind: Service
metadata:
```



```
name: cluster-vmstorage
namespace: kube-vm
spec:
  ports:
    - name: http
      port: 8482
      targetPort: 8482
    - name: vminsert
      port: 8400
      targetPort: 8400
    - name: vmselect
      port: 8401
      targetPort: 8401
  selector:
    app: vmstorage
```

应用配置：

```
kubectl apply -f cluster-vmstorage.yaml
```

4. 验证集群部署

- 检查所有组件是否运行正常：

```
kubectl get pods -n kube-vm
```

- 验证 `vminsert` 和 `vmselect` 是否成功连接到 `vmstorage`。

5. 配置数据采集和查询

采集数据

使用 `vmagent` 替换 Prometheus 作为采集器：

- **vmagent** 的主要功能：
 - 支持 Prometheus 的 `scrape_configs`。
 - 支持将数据写入多个存储后端。

查询数据

在 Grafana 中添加 VictoriaMetrics 数据源：

1. 地址设置为 `http://<vmselect-ip>:8481`
2. 测试连接并导入监控 Dashboard。

6. 高可用与扩展

- 添加更多 `vmstorage` 节点可扩展存储。





- 增加 `vminsert` 和 `vmselect` 节点以提高并发处理能力。
- 设置负载均衡器路由到健康节点。

7. 备份与恢复

备份

使用 `vmbackup` 从快照中备份数据：

1. 创建快照：

```
curl http://<vmstorage-ip>:8482/snapshot/create
```

2. 使用 `vmbackup` 归档快照数据。

恢复

使用 `vmrestore` 恢复数据到 `vmstorage` 节点。

如果需要更具体的调优建议或问题排查，请告诉我！



参考“一文搞懂 VictoriaMetrics 的使用-阳明的博客.pdf”，详细一步步教我怎么做，能比较出与 prometheus 的不同



以下是基于《一文搞懂 VictoriaMetrics 的使用-阳明的博客.pdf》文件 ，一步步教您如何部署 **VictoriaMetrics 集群模式** 并与 Prometheus 进行比较的详细指南：

一、VictoriaMetrics 与 Prometheus 的主要区别

1. 性能与扩展性

- **VictoriaMetrics**：支持水平扩展，集群模式下更适合处理高吞吐量和高基数数据，数据压缩更高效。
- **Prometheus**：主要用于单实例部署，高吞吐场景下性能可能不足。

2. 数据存储与压缩

- **VictoriaMetrics**：内存和存储占用更少，数据压缩率高。
- **Prometheus**：数据存储本地化，易受硬件限制。

3. 兼容性

- **VictoriaMetrics**：完全兼容 Prometheus API，可替换 Prometheus，支持更多数据协议。





- **Prometheus**：广泛应用于社区，支持标准的 PromQL 查询。

4. 操作复杂度

- **VictoriaMetrics**：单一可执行文件，配置简单；集群模式需要单独部署组件。
- **Prometheus**：单节点部署简单，但高可用和远程存储需要额外组件（如 Thanos 或 Cortex）。

二、部署 VictoriaMetrics 集群模式

1. 准备工作

- 确保 Kubernetes 集群可用。
- 准备用于 `PersistentVolume` 的存储。
- 安装 `kubect1` 和（可选）Helm 工具。

2. 部署 `vmstorage` 节点

`vmstorage` 负责存储数据并响应查询请求。

创建配置文件 `vmstorage.yaml`

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vmstorage
  namespace: kube-vm
spec:
  serviceName: cluster-vmstorage
  replicas: 2
  selector:
    matchLabels:
      app: vmstorage
  template:
    metadata:
      labels:
        app: vmstorage
    spec:
      containers:
        - name: vmstorage
          image: victoriametrics/vmstorage:v1.91.3-cluster
          args:
            - "--retentionPeriod=1"
            - "--storageDataPath=/storage"
          ports:
            - name: http
              containerPort: 8482
            - name: vminsert
              containerPort: 8400
            - name: vmselect
              containerPort: 8401
          volumeMounts:
            - name: storage
```



```
        mountPath: /storage
volumeClaimTemplates:
- metadata:
    name: storage
  spec:
    accessModes: ["ReadWriteOnce"]
    resources:
      requests:
        storage: 100Gi
```

应用配置

```
kubectl apply -f vmstorage.yaml
```

3. 部署 `vminsert` 节点

`vminsert` 负责接收数据并将其分发到 `vmstorage` 节点。

创建配置文件 `vminsert.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: kube-vm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
      - name: vminsert
        image: victoriametrics/vminsert:v1.91.3-cluster
        args:
          - "--storageNode=cluster-vmstorage:8400"
        ports:
          - name: http
            containerPort: 8480
```

应用配置

```
kubectl apply -f vminsert.yaml
```

4. 部署 `vmselect` 节点

`vmselect` 用于从 `vmstorage` 中查询数据。



创建配置文件 `vmselect.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmselect
  namespace: kube-vm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vmselect
  template:
    metadata:
      labels:
        app: vmselect
    spec:
      containers:
        - name: vmselect
          image: victoriametrics/vmselect:v1.91.3-cluster
          args:
            - "--storageNode=cluster-vmstorage:8401"
          ports:
            - name: http
              containerPort: 8481
```

应用配置

```
kubectl apply -f vmselect.yaml
```

5. 验证集群部署

```
kubectl get pods -n kube-vm
kubectl get svc -n kube-vm
```

- 确保 `vmstorage`、`vminsert` 和 `vmselect` 节点正常运行。
- 访问 `vmselect` 的 HTTP 接口：`http://<vmselect-ip>:8481/metrics`。

三、数据采集与查询

1. 替换 Prometheus 的数据采集

使用 `vmagent` 替换 Prometheus 的采集功能，保留 Prometheus 的 `scrape_configs`：

vmagent 配置示例

```
scrape_configs:
  - job_name: 'example'
    static_configs:
      - targets: ['localhost:9100']
```



2. 数据查询

在 Grafana 中配置 VictoriaMetrics 数据源：

- URL: `http://<vmselect-ip>:8481``
- 类型：Prometheus

3. 对比 Prometheus

- 数据压缩性能：VictoriaMetrics 数据存储效率高，占用更少。
- 资源消耗：VictoriaMetrics 内存和 CPU 消耗比 Prometheus 更低。
- 操作复杂度：VictoriaMetrics 集群模式需要更多配置，而 Prometheus 使用单实例更简单。

四、总结

通过上述部署，您可以观察 VictoriaMetrics 在以下方面的表现：

- 查询性能（高并发）
- 数据压缩率
- 高可用与水平扩展能力

如果需要更详细的调优或对比实验，欢迎进一步探讨！



怎麼設置查詢性能的實驗



为了评估 **VictoriaMetrics** 和 **Prometheus** 的查询性能，可以通过设计一个实验来模拟真实的高并发查询负载并测量其响应时间、吞吐量等关键性能指标。以下是详细的步骤：

1. 实验目标

比较 VictoriaMetrics 和 Prometheus 在以下方面的查询性能：

1. **响应时间**：每次查询完成所需时间。
2. **吞吐量**：每秒处理的查询数量（Queries Per Second, QPS）。
3. **资源占用**：CPU、内存的使用情况。
4. **查询准确性**：结果是否一致。

2. 实验准备

硬件与软件环境

1. 部署以下两种监控系统：



- **Prometheus**：单实例（或者加上 Thanos 远程存储）。
 - **VictoriaMetrics**：单节点或集群模式。
2. 确保硬件环境一致，避免因硬件差异导致实验结果失真。

测试工具

- **k6**：用于模拟高并发的 HTTP 查询负载。
- **wrk**：高性能 HTTP 压测工具。
- **Grafana**：查询可视化和数据分析。

测试数据

1. 使用 `node-exporter` 或其他模拟负载生成指标数据。
2. 采集至少 1 天以上的时序序列数据，确保数据量足够。

3. 实验步骤

步骤 1：配置测试场景

1. **数据准备**：
 - 确保 Prometheus 和 VictoriaMetrics 都采集相同的监控指标（如 `node_exporter` 的指标）。
 - 配置相同的 `scrape_interval`（如 15 秒）。
2. **查询模板**：选择以下常见的 PromQL 查询语句，覆盖简单到复杂的查询场景：

- 简单查询：

```
node_cpu_seconds_total{instance="localhost:9100", mode="idle"}
```

- 聚合查询：

```
avg(rate(node_cpu_seconds_total[1m]))
```

- 高基数查询：

```
sum by (instance)(rate(node_network_receive_bytes_total[5m]))
```

- 时间范围查询：

```
node_memory_MemAvailable_bytes[1d]
```

步骤 2：执行查询性能测试

使用工具 `k6` 或 `wrk` 模拟多并发用户进行查询：

1. **k6 脚本示例**：创建 `load-test.js` 文件：

```
import http from "k6/http";
import { check, sleep } from "k6";
```



```
export let options = {
  stages: [
    { duration: "1m", target: 50 }, // 50 并发用户
    { duration: "3m", target: 200 }, // 200 并发用户
    { duration: "1m", target: 0 }, // 停止
  ],
};

export default function () {
  let res = http.get("http://<prometheus-or-vm-url>/api/v1/query?query=avg(rate(node_cpu_seconds_total[1m]))");
  check(res, { "status is 200": (r) => r.status === 200 });
  sleep(1);
}
```

执行测试：

```
k6 run load-test.js
```

2. wrk 命令示例：

```
wrk -t4 -c200 -d3m "http://<prometheus-or-vm-url>/api/v1/query?query=avg(rate(node_cpu_seconds_total[1m]))"
```

步骤 3：记录性能指标

1. 响应时间：使用工具的报告。
2. 吞吐量：QPS（工具自动统计）。
3. 资源占用：
 - 使用 `htop` 或 Kubernetes 自带的监控工具（如 `kubectl top pods`）监控 CPU 和内存使用情况。
4. 查询准确性：
 - 比较 Prometheus 和 VictoriaMetrics 的查询结果，验证是否一致。

4. 数据分析

收集以下关键数据：

1. 平均响应时间、最大响应时间。
2. 最大 QPS。
3. CPU 和内存使用曲线。
4. 数据准确性（结果差异）。

结果可视化

1. 将测试数据导入 Grafana。
2. 创建 Dashboard，展示以下图表：
 - 响应时间曲线。
 - QPS 曲线。
 - CPU、内存占用情况。



5. 实验优化建议

VictoriaMetrics

- 通过增加 `vmselect`` 节点提升查询并发能力。
- 优化存储节点数量，平衡负载。
- 调整查询去重参数 `-dedup.minScrapeInterval``。

Prometheus

- 优化 `scrape_interval`` 和 `query.timeout``。
- 增加远程存储（如 Thanos）以减少单实例负载。

6. 结果解读

1. **响应时间**：VictoriaMetrics 应在复杂查询场景下表现更优。
2. **吞吐量**：VictoriaMetrics 集群模式支持更高并发。
3. **资源占用**：VictoriaMetrics 对内存、CPU的占用通常低于 Prometheus。
4. **准确性**：结果差异说明可能存在查询语义或时间对齐问题。

通过以上实验，您可以清晰地比较 **VictoriaMetrics** 和 **Prometheus** 在查询性能、资源利用率等方面的优劣。需要更多优化或调整建议，欢迎进一步讨论！



怎麼設置評估比較 VictoriaMetrics 和 Prometheus 在数据压缩率的實驗



评估 VictoriaMetrics 和 Prometheus 数据压缩率的实验设计

数据压缩率是衡量时间序列数据库在存储同样数量的数据时，其存储空间利用效率的指标。以下是详细实验设计，用于比较 **VictoriaMetrics** 和 **Prometheus** 的数据压缩率。

1. 实验目标

评估 VictoriaMetrics 和 Prometheus 在相同条件下存储相同时间序列数据时的压缩率差异。

2. 实验准备

硬件环境



- 相同的服务器硬件或云环境，确保公平对比。
- 每个系统的存储路径应在单独的磁盘或目录中，以便精确测量。

软件环境

- **Prometheus** 和 **VictoriaMetrics** 单节点部署。
- **数据源**：使用 `node-exporter` 或生成器工具（如 `prometheus-metrics-generator`）。

实验工具

- `du`：用于统计存储目录的大小。
- `curl` 或 `Prometheus` 的 API，用于验证采集数据一致性。

实验数据

- 模拟真实场景，使用 `node-exporter` 采集 CPU、内存、磁盘等常见指标。
- 采样间隔：15 秒。
- 数据采集周期：1 天以上。

3. 实验步骤

步骤 1：部署两个系统

1. **部署 Prometheus** 配置 `prometheus.yaml`：

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "node-exporter"
    static_configs:
      - targets: ["<node-exporter-ip>:9100"]
```

启动命令：

```
prometheus --config.file=prometheus.yaml --storage.tsdb.path=/data/prometheus
```

2. **部署 VictoriaMetrics** 配置 `vmagent` 的 `scrape_configs`，与 Prometheus 配置相同：

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "node-exporter"
    static_configs:
      - targets: ["<node-exporter-ip>:9100"]
```

启动命令：



```
vmagent -promscrape.config=/etc/vmagent.yaml -remoteWrite.url=http://localhost:8428/api/v1/write
victoriametrics -storageDataPath=/data/victoriametrics -retentionPeriod=1h
```

步骤 2：采集相同的数据

- 确保两者采集的时间序列指标相同。
- 采集时间：运行 1 天或更长时间。

步骤 3：测量存储大小

在采集完成后，分别检查存储目录的大小：

1. Prometheus 数据大小

```
du -sh /data/prometheus
```

2. VictoriaMetrics 数据大小

```
du -sh /data/victoriametrics
```

步骤 4：校验数据一致性

- 使用 `PromQL` 查询相同指标，确保两者数据一致。
- 示例查询：

```
curl "http://<prometheus-ip>:9090/api/v1/query?query=rate(node_cpu_seconds_total[1m])"
curl "http://<victoriametrics-ip>:8428/api/v1/query?query=rate(node_cpu_seconds_total[1m])"
```

4. 数据分析

计算压缩率

压缩率的计算公式： $\text{Compression Ratio} = \frac{\text{Prometheus Data Size}}{\text{VictoriaMetrics Data Size}}$

结果对比

1. 如果 VictoriaMetrics 的数据目录明显小于 Prometheus，则表明其压缩率更高。
2. 注意对比以下指标：
 - 总存储占用
 - 数据一致性
 - 数据查询时间（验证是否影响查询性能）

5. 优化建议

VictoriaMetrics

- 调整 `retentionPeriod` 参数控制数据保留时间。
- 调整 `storageDataPath` 目录的存储配置，确保性能最佳。

Prometheus



- 使用 `compaction` 优化存储空间。
- 结合 Thanos 等远程存储解决方案优化长时间存储。

6. 扩展实验

影响因素测试

1. **高基数数据测试：**
 - 增加时间序列的数量，例如通过模拟大量虚拟指标。
2. **高频率采样测试：**
 - 减少采样间隔到 5 秒或 1 秒，观察存储占用的变化。
3. **长时间存储测试：**
 - 延长存储周期，验证压缩率是否随时间变化。

查询性能对比

- 在存储大量数据后，测试 VictoriaMetrics 和 Prometheus 的查询性能。
- 使用复杂的聚合查询和时间范围查询进行比较。

通过上述实验，您可以清晰地量化 **VictoriaMetrics** 和 **Prometheus** 在数据压缩率和资源利用率上的差异。若需更多优化建议，欢迎进一步讨论！



怎麼設置評估比較 VictoriaMetrics 和 Prometheus 在高可用与水平扩展能力方面的實驗



为了比较 **VictoriaMetrics** 和 **Prometheus** 在 **高可用性** 和 **水平扩展能力** 方面的表现，可以设计以下实验：

1. 实验目标

1. **高可用性：**
 - 测试在组件故障时系统的可用性。
 - 验证数据丢失情况和查询结果的完整性。
2. **水平扩展能力：**
 - 测试增加组件节点后的数据采集和查询吞吐量。
 - 测量系统性能随节点数量增加的线性扩展性。



2. 实验准备

硬件与软件环境

- 部署在相同的 Kubernetes 集群或裸机环境中。
- 每种系统使用相同的硬件配置。
- 确保网络环境稳定。

系统架构

- **VictoriaMetrics**：部署集群模式，包括 `vmstorage`、`vminsert` 和 `vmselect``。
- **Prometheus**：
 - 使用单实例（基础对比）。
 - 使用 Thanos（模拟分布式存储的高可用场景）。

测试工具

- **k6** 或 **wrk**：用于模拟并发查询。
- **node-exporter** 或模拟负载生成器：用于生成监控数据。

数据类型

- 使用 `node-exporter` 收集 CPU、内存、磁盘等常规指标。`
- 数据采样间隔：15 秒。

3. 实验设计

A. 高可用性测试

步骤 1：部署高可用架构

1. **VictoriaMetrics**：
 - `vmstorage`：至少 2 个节点。`
 - `vminsert` 和 `vmselect`：各 2 个节点，设置负载均衡。`
2. **Prometheus**：
 - 单实例。
 - 加入 Thanos Sidecar，连接对象存储（如 MinIO）以提供高可用存储。

步骤 2：测试系统故障恢复能力

1. 故障模拟：
 - 停止一个 `vmstorage` 节点：`

```
kubectl scale statefulset vmstorage --replicas=1 -n kube-vm
```

- 停止一个 `vminsert` 或 `vmselect` 节点：`

```
kubectl scale deployment vminsert --replicas=1 -n kube-vm
```

2. 验证测试：



- **VictoriaMetrics :**
 - 查询数据是否仍然可用（通过其他存活节点）。
 - 验证数据一致性（确保不丢失）。
- **Prometheus :**
 - 停止主 Prometheus 实例，检查是否可以通过 Thanos 查询历史数据。

```
kubectl scale deployment prometheus --replicas=0
```

3. 实验指标：

- 查询响应时间和结果正确性。
- 数据丢失率。

B. 水平扩展能力测试

步骤 1：初始部署

1. 配置初始节点：

- **VictoriaMetrics :**
 - 1 个 `vmstorage` 节点。
 - 1 个 `vminsert` 和 `vmselect` 节点。
- **Prometheus :**
 - 单实例。

2. 使用 `node-exporter` 提供相同的数据源。

步骤 2：逐步增加节点

1. 增加 VictoriaMetrics 节点：

- 添加 `vmstorage` 节点：

```
kubectl scale statefulset vmstorage --replicas=2 -n kube-vm
```

- 添加 `vminsert` 和 `vmselect` 节点：

```
kubectl scale deployment vminsert --replicas=2 -n kube-vm  
kubectl scale deployment vmselect --replicas=2 -n kube-vm
```

2. 增加 Prometheus 实例：

- 部署联邦架构，多个 Prometheus 采集不同任务，并通过 Thanos 聚合。

步骤 3：测试扩展性能

1. 模拟高数据采集速率：

- 使用 `prometheus-metrics-generator` 生成大规模时间序列数据。



- 目标：每秒 100k 数据点。

2. 模拟高并发查询：

- 使用 `k6` 模拟 500 并发查询，逐步增加负载。
- 查询样例：

```
http://<system-url>/api/v1/query?query=rate(node_cpu_seconds_total[1m])
```

步骤 4：收集实验数据

- 查询吞吐量（QPS）。
- 平均查询响应时间。
- CPU、内存使用率。
- 存储利用率。

4. 数据分析

高可用性

1. 数据完整性：

- 检查在节点故障期间是否出现数据丢失或查询结果不一致。
- VictoriaMetrics 应通过多节点存储保证无数据丢失，而 Prometheus 需依赖对象存储。

2. 查询性能：

- VictoriaMetrics 在单节点故障时的查询性能应优于 Prometheus 联邦架构。

水平扩展能力

1. 扩展线性度：

- 吞吐量（QPS）是否随着节点数量的增加接近线性增长。
- VictoriaMetrics 的 `vmselect` 和 `vminsert` 节点应能够分担更多负载。

2. 资源利用率：

- 比较扩展后的 CPU 和内存利用率。
- VictoriaMetrics 的资源使用通常低于 Prometheus 联邦架构。

5. 实验扩展

影响因素测试

- **高基数数据测试**：增加时间序列数量（模拟 1 万个指标）。
- **高频数据采集**：减少采样间隔（如 5 秒）。
- **长时间存储测试**：延长数据保留周期（如 1 年）。





优化建议

1. VictoriaMetrics :

- 添加更多 `vmstorage` 节点，提高数据存储和查询能力。
- 调整 `-dedup.minScrapeInterval` 参数优化去重性能。

2. Prometheus :

- 使用对象存储（如 S3 或 MinIO）替代本地存储，提升持久化能力。
- 优化联邦架构中的查询聚合逻辑。

通过上述实验，您可以全面评估 **VictoriaMetrics** 和 **Prometheus** 在高可用性和水平扩展能力上的表现差异。需要进一步细化或实现细节，随时联系！



詳細說明VictoriaMetrics vmagent用法



VictoriaMetrics `vmagent` 用法详解

`vmagent` 是 VictoriaMetrics 的数据采集代理，类似于 Prometheus 的功能，负责从各种数据源中抓取监控指标，并通过远程写入功能将数据发送到后端存储（如 VictoriaMetrics）。

以下是 `vmagent` 的功能、配置和使用方法的详细说明。

1. `vmagent` 的主要功能

1. 替代 Prometheus 的数据抓取功能：

- 支持 Prometheus 的 `scrape_configs`，兼容 Prometheus 的配置格式。
- 可直接与 Kubernetes、Consul 等服务发现系统集成。

2. 多目标写入：

- 支持将数据发送到多个后端存储。
- 支持 Prometheus `remote_write` 协议。

3. 资源效率高：

- 比 Prometheus 占用更少的内存、CPU 和磁盘 IO。
- 内置缓存，保证数据采集的可靠性。

4. 灵活的数据过滤与修改：

- 支持 Prometheus 风格的 `relabel_configs`。
- 可对抓取的数据进行过滤、修改或动态调整标签。





5. 高可用性支持：

- 可部署多个 `vmagent` 实例，以实现数据的冗余和高可用。

2. `vmagent` 的部署方式

A. 使用 Docker 部署

```
docker run -d --name vmagent \
-v /path/to/vmagent-config.yaml:/etc/vmagent-config.yaml \
-p 8429:8429 \
victoriametrics/vmagent:latest \
-promscrape.config=/etc/vmagent-config.yaml \
-remoteWrite.url=http://<victoriametrics-url>:8428/api/v1/write
```

B. 使用 Kubernetes 部署

创建配置文件

`vmagent.yaml`：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmagent
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
      labels:
        app: vmagent
    spec:
      containers:
        - name: vmagent
          image: victoriametrics/vmagent:v1.91.3
          args:
            - "-promscrape.config=/etc/vmagent/vmagent-config.yaml"
            - "-remoteWrite.url=http://victoria-metrics:8428/api/v1/write"
          volumeMounts:
            - name: vmagent-config
              mountPath: /etc/vmagent
      volumes:
        - name: vmagent-config
          configMap:
            name: vmagent-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: vmagent-config
  namespace: monitoring
data:
  vmagent-config.yaml: |
```



```
global:
  scrape_interval: 15s
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'node-exporter'
    static_configs:
      - targets: ['localhost:9100']
```

应用配置

```
kubectl apply -f vmagent.yaml
```

3. `vmagent` 的关键配置项

A. `-promscrape.config`

指定 Prometheus 格式的抓取配置文件。

配置示例

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'node-exporter'
    static_configs:
      - targets:
        - 'localhost:9100'

  - job_name: 'kubernetes-pods'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: my-app
```

B. `-remoteWrite.url`

配置数据写入目标。可以指定多个 `remoteWrite` 地址。

示例

```
-remoteWrite.url=http://victoriametrics:8428/api/v1/write
-remoteWrite.url=http://prometheus-remote-storage:9090/api/v1/write
```

C. 数据过滤与重标记 (`relabel_configs`)

用于动态调整抓取数据的标签。

配置示例



```
scrape_configs:
  - job_name: 'example'
    static_configs:
      - targets: ['localhost:8080']
    relabel_configs:
      - source_labels: ['__address__']
        target_label: 'instance'
        replacement: 'example-instance'
      - source_labels: ['__name__']
        regex: 'http_requests_total'
        action: drop
```

4. `vmagent` 高级特性

A. 支持多种数据协议

`vmagent` 可抓取以下协议的数据：

- Prometheus 格式 (`remote_write`)
- InfluxDB 格式
- OpenTSDB 格式
- Graphite 格式
- JSON Lines 和 CSV 格式

B. 数据冗余与高可用

通过部署多个 `vmagent` 实例，并配置多个 `remoteWrite` 地址实现冗余数据写入：

```
-remoteWrite.url=http://victoria-metrics:8428/api/v1/write
-remoteWrite.url=http://backup-victoria-metrics:8428/api/v1/write
```

C. 缓存与本地存储

`vmagent` 支持在数据写入目标不可用时，缓存数据到本地磁盘：

```
-remoteWrite.tmpDataPath=/tmp/vmagent-data
```

5. 监控 `vmagent`

内置指标

`vmagent` 提供内置的监控指标，可通过 `/metrics` 接口查看：

```
curl http://<vmagent-ip>:8429/metrics
```

常见监控项

- `vm_promscrape_scrapes_total`：成功抓取次数。
- `vm_promscrape_targets`：当前抓取目标数量。
- `vm_remote_write_rows_total`：写入目标的样本总数。



6. 实用场景

场景 1：替代 Prometheus 的抓取功能

使用 `vmagent` 替代 Prometheus 的抓取功能，将数据写入 VictoriaMetrics 后端：

- 优势：占用资源更少，抓取效率更高。

场景 2：数据冗余存储

将数据同时写入多个 VictoriaMetrics 或 Prometheus 存储实例，确保数据安全性。

场景 3：复杂数据过滤

通过 `relabel_configs` 对采集数据进行标签过滤、修改，精简数据量。

场景 4：整合多种协议数据源

同时抓取来自 Prometheus、InfluxDB 和 OpenTSDB 的指标数据。

通过以上配置和功能介绍，您可以高效地使用 `vmagent` 替代或增强 Prometheus 的数据抓取能力。若需要更多细化配置或场景应用指导，请随时提出！



怎麼部署VictoriaMetrics多節點



部署 **VictoriaMetrics 多节点（集群模式）** 是为了满足高并发、高吞吐量场景的需求。以下是详细的多节点部署指南：

1. 集群架构概述

VictoriaMetrics 集群模式包含以下组件：

1. `vmstorage`：
 - 负责存储数据。
 - 数据根据标签哈希分布到不同节点。
2. `vminsert`：
 - 负责接收采集数据，将其分发到 `vmstorage` 节点。
3. `vmselect`：
 - 负责查询数据，从 `vmstorage` 节点获取所需结果。

2. 部署前准备



硬件需求

- **CPU**：根据数据量和并发查询需求，建议 `4+ cores`。
- **内存**：至少 8GB，推荐 16GB 或更多。
- **存储**：SSD 推荐，确保低延迟、高 IOPS。
- **网络**：低延迟、高带宽网络连接。

软件需求

- Kubernetes 集群（推荐），或者 Docker 环境。
- Kubernetes `kubectl` 工具。

3. 多节点部署步骤

步骤 1：创建命名空间

创建一个独立的命名空间用于部署组件：

```
kubectl create namespace vm-cluster
```

步骤 2：部署 `vmstorage`

`vmstorage` 是有状态的组件，使用 `StatefulSet` 部署。

配置文件 `vmstorage.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: cluster-vmstorage
  namespace: vm-cluster
spec:
  clusterIP: None
  ports:
    - port: 8482
      name: http
    - port: 8400
      name: vminsert
    - port: 8401
      name: vmselect
  selector:
    app: vmstorage
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vmstorage
  namespace: vm-cluster
spec:
  serviceName: cluster-vmstorage
  replicas: 3 # 部署 3 个副本
  selector:
    matchLabels:
      app: vmstorage
  template:
    metadata:
```



```
labels:
  app: vmstorage
spec:
  containers:
    - name: vmstorage
      image: victoriametrics/vmstorage:v1.91.3-cluster
      args:
        - "--retentionPeriod=1" # 数据保留周期 (1年)
        - "--storageDataPath=/storage"
      ports:
        - name: http
          containerPort: 8482
        - name: vminsert
          containerPort: 8400
        - name: vmselect
          containerPort: 8401
      volumeMounts:
        - name: storage
          mountPath: /storage
  volumeClaimTemplates:
    - metadata:
        name: storage
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 100Gi # 每个节点分配 100Gi 存储
```

部署 `vmstorage`

```
kubectl apply -f vmstorage.yaml
```

步骤 3：部署 `vminsert`

`vminsert` 将采集的数据分发到多个 `vmstorage` 节点。

配置文件 `vminsert.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: vminsert
  namespace: vm-cluster
spec:
  ports:
    - port: 8480
      name: http
  selector:
    app: vminsert
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: vm-cluster
spec:
```



```
replicas: 2
selector:
  matchLabels:
    app: vminsert
template:
  metadata:
    labels:
      app: vminsert
  spec:
    containers:
      - name: vminsert
        image: victoriametrics/vminsert:v1.91.3-cluster
        args:
          - "--storageNode=cluster-vmstorage:8400" # 连接到 vmstorage
        ports:
          - name: http
            containerPort: 8480
```

部署 `vminsert`

```
kubectl apply -f vminsert.yaml
```

步骤 4：部署 `vmselect`

`vmselect` 负责查询数据，从 `vmstorage` 获取结果。

配置文件 `vmselect.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: vmselect
  namespace: vm-cluster
spec:
  ports:
    - port: 8481
      name: http
  selector:
    app: vmselect
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmselect
  namespace: vm-cluster
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vmselect
  template:
    metadata:
      labels:
        app: vmselect
    spec:
      containers:
```



```
- name: vmselect
  image: victoriametrics/vmselect:v1.91.3-cluster
  args:
    - "--storageNode=cluster-vmstorage:8401" # 连接到 vmstorage
  ports:
    - name: http
      containerPort: 8481
```

部署 `vmselect`

```
kubectl apply -f vmselect.yaml
```

4. 验证集群部署

检查 Pods 状态

```
kubectl get pods -n vm-cluster
```

验证服务

通过端口转发访问组件接口：

```
kubectl port-forward svc/vmselect 8481:8481 -n vm-cluster
```

然后访问查询接口：

```
curl http://localhost:8481/metrics
```

5. 优化与扩展

水平扩展

1. **扩展存储节点**：增加 `vmstorage` 的副本数量：

```
kubectl scale statefulset vmstorage --replicas=5 -n vm-cluster
```

2. **扩展查询节点**：增加 `vmselect` 的副本数量：

```
kubectl scale deployment vmselect --replicas=4 -n vm-cluster
```

监控与故障排查

1. **监控集群性能**：

- `vm_promscrape_targets`：当前抓取的目标数量。
- `vm_remote_write_rows_total`：远程写入的样本总数。

2. **日志检查**：查看组件日志排查问题：



```
kubect1 logs -l app=vmselect -n vm-cluster
```

6. 总结

VictoriaMetrics 多节点集群模式提供了强大的水平扩展能力，可处理大规模时间序列数据。通过分离组件职责（`vmstorage`、`vminsert`、`vmselect`），实现了高性能和高可用性。按照以上步骤，您可以成功部署并运行多节点集群，同时根据需求调整配置以满足业务需求。

如果需要更多帮助或优化建议，请随时联系！



不使用k8s怎麼部署，詳細說明



以下是详细的 **VictoriaMetrics 多节点集群模式** 部署指南，**不使用 Kubernetes**，而是基于裸机或虚拟机环境直接部署。

1. 集群架构概述

VictoriaMetrics 集群模式包含以下组件：

1. `vmstorage``：负责存储数据。
2. `vminsert``：接收采集数据，将其分发到 `vmstorage`` 节点。
3. `vmselect``：处理查询，从 `vmstorage`` 获取结果。

2. 部署前准备

硬件需求

- **CPU**：根据数据量和查询需求，建议 `4+ cores``。
- **内存**：16GB 或更多。
- **存储**：SSD（推荐），确保低延迟和高 IOPS。
- **网络**：确保节点间网络低延迟、高带宽。

操作系统需求

- Linux 发行版：建议 Ubuntu 20.04 或 CentOS 7+。
- 安装以下工具：
 - `wget`` 或 `curl``
 - `systemctl``（用于服务管理）



3. 下载 VictoriaMetrics 可执行文件

下载最新版本的 **VictoriaMetrics 集群模式** 可执行文件：

步骤

1. 前往 [VictoriaMetrics Releases](#)。
2. 下载以下组件的最新版本：

- ``vmstorage``
- ``vminsert``
- ``vmselect``

例如：

```
wget https://github.com/VictoriaMetrics/VictoriaMetrics/releases/download/v1.91.3/vmstorage-linux-amd64-v1.91.3.tar.gz
wget https://github.com/VictoriaMetrics/VictoriaMetrics/releases/download/v1.91.3/vminsert-linux-amd64-v1.91.3.tar.gz
wget https://github.com/VictoriaMetrics/VictoriaMetrics/releases/download/v1.91.3/vmselect-linux-amd64-v1.91.3.tar.gz
```

3. 解压缩文件：

```
tar -xzf vmstorage-linux-amd64-v1.91.3.tar.gz
tar -xzf vminsert-linux-amd64-v1.91.3.tar.gz
tar -xzf vmselect-linux-amd64-v1.91.3.tar.gz
```

4. 移动可执行文件到系统路径：

```
sudo mv vmstorage-linux-amd64 /usr/local/bin/vmstorage
sudo mv vminsert-linux-amd64 /usr/local/bin/vminsert
sudo mv vmselect-linux-amd64 /usr/local/bin/vmselect
```

4. 配置和启动各组件

A. 部署 ``vmstorage``

创建存储目录

```
sudo mkdir -p /var/lib/vmstorage
sudo chown -R $USER:$USER /var/lib/vmstorage
```

启动命令

启动 ``vmstorage``，监听默认端口（8482）：

```
vmstorage \
  -retentionPeriod=1y \
  -storageDataPath=/var/lib/vmstorage \
  -httpListenAddr=:8482 \
  -vminsertAddr=:8400 \
  -vmselectAddr=:8401
```

后台运行（推荐使用 **Systemd**）



创建 Systemd 服务文件：

```
sudo nano /etc/systemd/system/vmstorage.service
```

服务文件内容：

```
[Unit]
Description=VictoriaMetrics Storage Node
After=network.target

[Service]
ExecStart=/usr/local/bin/vmstorage \
  -retentionPeriod=1y \
  -storageDataPath=/var/lib/vmstorage \
  -httpListenAddr=:8482 \
  -vminsertAddr=:8400 \
  -vmselectAddr=:8401
Restart=always
User=$USER

[Install]
WantedBy=multi-user.target
```

启用并启动服务：

```
sudo systemctl daemon-reload
sudo systemctl enable vmstorage
sudo systemctl start vmstorage
```

B. 部署 `vminsert`

启动命令

启动 `vminsert`，连接到 `vmstorage`：

```
vminsert \
  -storageNode=127.0.0.1:8400 \
  -httpListenAddr=:8480
```

后台运行 (Systemd 配置)

```
sudo nano /etc/systemd/system/vminsert.service
```

服务文件内容：

```
[Unit]
Description=VictoriaMetrics Insert Node
After=network.target

[Service]
ExecStart=/usr/local/bin/vminsert \
  -storageNode=127.0.0.1:8400 \
```



```
-httpListenAddr=:8480
Restart=always
User=$USER

[Install]
WantedBy=multi-user.target
```

启用并启动服务：

```
sudo systemctl daemon-reload
sudo systemctl enable vminsert
sudo systemctl start vminsert
```

C. 部署 `vmselect`

启动命令

启动 `vmselect`，连接到 `vmstorage`：

```
vmselect \
  -storageNode=127.0.0.1:8401 \
  -httpListenAddr=:8481
```

后台运行 (Systemd 配置)

```
sudo nano /etc/systemd/system/vmselect.service
```

服务文件内容：

```
[Unit]
Description=VictoriaMetrics Select Node
After=network.target

[Service]
ExecStart=/usr/local/bin/vmselect \
  -storageNode=127.0.0.1:8401 \
  -httpListenAddr=:8481
Restart=always
User=$USER

[Install]
WantedBy=multi-user.target
```

启用并启动服务：

```
sudo systemctl daemon-reload
sudo systemctl enable vmselect
sudo systemctl start vmselect
```



5. 验证部署

检查服务状态

```
sudo systemctl status vmstorage
sudo systemctl status vminsert
sudo systemctl status vmselect
```

测试接口

1. 检查 `vmstorage` 是否启动：

```
curl http://127.0.0.1:8482/metrics
```

2. 检查 `vminsert` 是否正常写入：

```
curl http://127.0.0.1:8480/metrics
```

3. 检查 `vmselect` 查询是否正常：

```
curl http://127.0.0.1:8481/metrics
```

6. 高可用与扩展

扩展存储节点

增加多个 `vmstorage` 节点，在 `vminsert` 和 `vmselect` 中添加存储节点地址：

```
vminsert \
  -storageNode=127.0.0.1:8400,127.0.0.2:8400
vmselect \
  -storageNode=127.0.0.1:8401,127.0.0.2:8401
```

负载均衡

使用 Nginx 或 HAProxy 对 `vminsert` 和 `vmselect` 进行负载均衡。

数据冗余

通过多个 `vminsert` 和 `vmselect` 实例实现数据冗余。

7. 监控与调试

监控指标

VictoriaMetrics 提供内置的 `/metrics` 接口，可以通过 Prometheus 或 Grafana 监控以下关键指标：

- `vm_http_requests_total`：HTTP 请求总数。
- `vm_rows_inserted_total`：写入的样本行数。
- `vm_rows_queried_total`：查询的样本行数。



日志检查

如果遇到问题，查看日志：

```
journalctl -u vmstorage
journalctl -u vminsert
journalctl -u vmselect
```

通过以上步骤，您可以在裸机或虚拟机环境下成功部署 **VictoriaMetrics 多节点集群模式**，并根据业务需求进行扩展和优化。如需更多帮助，欢迎随时咨询！



詳細說明怎麼測試部署



测试部署后的 **VictoriaMetrics 多节点集群模式** 是否正常工作，是确保系统高效稳定运行的关键步骤。以下是详细的测试方法：

1. 测试目标

1. 验证各组件（`vmstorage`、`vminsert`、`vmselect`）的功能是否正常。
2. 测试数据采集、存储和查询的完整性与性能。
3. 评估集群的稳定性、容错能力和扩展性。

2. 基础功能测试

A. 验证各组件运行状态

1. 检查组件是否在运行：

```
systemctl status vmstorage
systemctl status vminsert
systemctl status vmselect
```

2. 测试组件的 HTTP 接口：

- `vmstorage`：

```
curl http://127.0.0.1:8482/metrics
```

输出应该包含系统监控指标，如 `vm_rows_inserted_total`。

- `vminsert`：





```
curl http://127.0.0.1:8480/metrics
```

- `\vmselect`` :

```
curl http://127.0.0.1:8481/metrics
```

B. 测试数据写入

使用 `\curl`` 或 Prometheus 的 `\remote_write`` 接口模拟数据写入：

```
curl -X POST "http://127.0.0.1:8480/api/v1/write" \  
-d @test_data.snappy
```

示例数据 (`test_data.snappy`)：

```
{"streams":[{"labels":{"__name__":"test_metric","instance":"localhost"},"samples":[{"value":1.23,"timestamp":1672531200000}
```

C. 测试数据查询

使用 `\api/v1/query`` 接口查询写入的数据：

```
curl "http://127.0.0.1:8481/api/v1/query?query=test_metric"
```

验证返回结果是否正确，如：

```
{  
  "status": "success",  
  "data": {  
    "resultType": "vector",  
    "result": [  
      {  
        "metric": {  
          "__name__": "test_metric",  
          "instance": "localhost"  
        },  
        "value": [1672531200000, "1.23"]  
      }  
    ]  
  }  
}
```

3. 性能测试

A. 数据写入性能

1. 使用 [prometheus-metrics-generator](#) 生成大量数据。
2. 将数据通过 `\remote_write`` 接口写入 `\vminsert``：

```
prometheus-metrics-generator -remoteWrite.url=http://127.0.0.1:8480/api/v1/write -series=100000 -interval=1s
```

3. 观察 `\vmstorage`` 的写入性能指标：



```
curl http://127.0.0.1:8482/metrics | grep "vm_rows_inserted_total"
```

B. 数据查询性能

1. 模拟高并发查询：使用 `wrk` 或 `k6` 测试 `/api/v1/query` 接口的查询性能。

```
wrk -t4 -c100 -d30s "http://127.0.0.1:8481/api/v1/query?query=test_metric"
```

参数说明：

- `-t4`：4 个线程。
 - `-c100`：100 个并发连接。
 - `-d30s`：运行 30 秒。
2. 分析性能指标：
 - 平均响应时间。
 - 最大吞吐量 (QPS)。

C. 系统资源占用

通过 `htop` 或 `top` 监控 CPU 和内存占用情况，检查是否存在资源瓶颈。

4. 数据完整性测试

A. 连续写入与查询

1. 配置一个 Prometheus 实例，将其 `remote_write` 设置为 VictoriaMetrics 的 `vminsert`：

```
remote_write:  
- url: "http://127.0.0.1:8480/api/v1/write"
```

2. 配置 `node-exporter` 或生成指标数据，采集一段时间。
3. 使用以下查询验证数据是否完整：

```
curl "http://127.0.0.1:8481/api/v1/query?query=rate(node_cpu_seconds_total[5m])"
```

B. 数据一致性

1. 在写入多个 `vmstorage` 节点时，验证相同查询结果是否一致。
2. 模拟节点故障：
 - 停止一个 `vmstorage` 节点：

```
systemctl stop vmstorage
```

- 查询结果是否受影响。



5. 高可用性测试

A. 故障恢复

1. 停止一个 `vminsert` 或 `vmselect` 节点：

```
systemctl stop vminsert
```

2. 验证其他节点是否继续提供写入和查询服务。

B. 数据冗余测试

1. 配置多个 `vminsert` 和 `vmselect` 节点。
2. 在负载均衡器后测试数据写入和查询，观察服务是否受影响。

6. 水平扩展测试

A. 添加新节点

1. 增加 `vmstorage` 节点：

```
vmstorage -retentionPeriod=1y -storageDataPath=/var/lib/vmstorage2 -httpListenAddr=:8483 -vminsertAddr=:8402 -vmselectA
```

2. 更新 `vminsert` 和 `vmselect` 配置，添加新节点：

```
vminsert -storageNode=127.0.0.1:8400,127.0.0.1:8402  
vmselect -storageNode=127.0.0.1:8401,127.0.0.1:8403
```

B. 性能对比

1. 写入和查询性能是否随节点增加而提升。
2. 检查新节点是否正常接收和处理数据。

7. 压力测试

A. 极端场景

1. 写入高频数据（1 秒 100k 数据点）：

```
prometheus-metrics-generator -remoteWrite.url=http://127.0.0.1:8480/api/v1/write -series=100000 -interval=1s
```

2. 模拟高并发查询（1000 并发）：

```
wrk -t10 -c1000 -d1m "http://127.0.0.1:8481/api/v1/query?query=test_metric"
```

B. 系统稳定性

1. 监控资源使用（CPU、内存、磁盘 IO）。
2. 检查日志是否有错误输出：





```
journalctl -u vmstorage
journalctl -u vminsert
journalctl -u vmselect
```

8. 总结

通过以上测试，您可以验证 VictoriaMetrics 集群模式的部署是否成功，评估其性能、稳定性和高可用性。如果需要更具体的测试脚本或优化建议，请进一步告知！



詳細此多節點怎麼跟vmagent測試



以下是详细的 **VictoriaMetrics 多节点集群模式** 如何与 **vmagent** 配合测试的指南：

1. 测试目标

1. 验证 `vmagent` 是否能正确抓取数据并发送到 VictoriaMetrics 多节点集群。
2. 测试数据写入的性能、完整性和高可用性。
3. 验证 `vmagent` 的抓取配置和负载能力。

2. 环境准备

A. 配置 VictoriaMetrics 集群

确保多节点集群的组件已经正确部署：

- `vmstorage`：存储节点。
- `vminsert`：写入节点。
- `vmselect`：查询节点。

测试环境中的端口示例：

- `vmstorage`：`8400`（vminsert 数据写入）和 `8401`（vmselect 查询）。
- `vminsert`：`8480`
- `vmselect`：`8481`

B. 下载并运行 vmagent

从 [VictoriaMetrics Releases](https://github.com/VictoriaMetrics/VictoriaMetrics/releases) 下载 `vmagent`：

```
wget https://github.com/VictoriaMetrics/VictoriaMetrics/releases/download/v1.91.3/vmagent-linux-amd64-v1.91.3.tar.gz
tar -xzvf vmagent-linux-amd64-v1.91.3.tar.gz
sudo mv vmagent-linux-amd64 /usr/local/bin/vmagent
```





创建数据目录：

```
sudo mkdir -p /etc/vmagent
sudo chown -R $USER:$USER /etc/vmagent
```

3. 配置 `vmagent`

A. `scrape_configs` 抓取配置

创建 `vmagent-config.yaml` 文件：

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s

scrape_configs:
  # 抓取 node-exporter 数据
  - job_name: "node-exporter"
    static_configs:
      - targets:
          - "localhost:9100" # 替换为您的 node-exporter 地址

  # 抓取自定义指标数据
  - job_name: "custom-metrics"
    static_configs:
      - targets:
          - "localhost:8080"

relabel_configs:
  - source_labels: [__address__]
    target_label: instance
    replacement: custom-instance
```

将配置文件放到 `/etc/vmagent/vmagent-config.yaml`。

B. 远程写入配置

启动 `vmagent` 时指定 `remote_write` 目标，指向 `vminsert`：

```
vmagent \
-promscrape.config=/etc/vmagent/vmagent-config.yaml \
-remoteWrite.url=http://127.0.0.1:8480/api/v1/write
```

支持同时写入多个目标（例如主集群和备份集群）：

```
vmagent \
-promscrape.config=/etc/vmagent/vmagent-config.yaml \
-remoteWrite.url=http://127.0.0.1:8480/api/v1/write \
-remoteWrite.url=http://backup-vminsert:8480/api/v1/write
```



4. 测试流程

A. 验证数据抓取

1. 检查 `vmagent` 是否正确抓取目标：

```
curl http://127.0.0.1:8429/targets
```

输出应显示抓取的所有目标及其状态：

```
{
  "activeTargets": [
    {
      "discoveredLabels": {
        "__address__": "localhost:9100",
        "job": "node-exporter"
      },
      "labels": {
        "instance": "localhost:9100",
        "job": "node-exporter"
      },
      "lastError": "",
      "lastScrape": "2025-01-07T10:00:00Z",
      "health": "up"
    }
  ]
}
```

2. 监控抓取数据的样本数：

```
curl http://127.0.0.1:8429/metrics | grep "vm_promscrape_targets"
```

B. 验证数据写入

1. 检查写入状态

- 查看 `vminsert` 是否收到数据：

```
curl http://127.0.0.1:8480/metrics | grep "vm_rows_inserted_total"
```

- 结果应显示写入的样本数：

```
vm_rows_inserted_total 1000
```

2. 检查存储节点状态

- 查看 `vmstorage` 是否正确存储数据：

```
curl http://127.0.0.1:8482/metrics | grep "vm_rows_inserted_total"
```



C. 测试数据查询

1. 使用 `\vmselect`` 查询写入的数据：

```
curl "http://127.0.0.1:8481/api/v1/query?query=rate(node_cpu_seconds_total[1m])"
```

2. 在 Grafana 配置数据源：

- 数据源类型：``Prometheus``
- URL：``http://127.0.0.1:8481``

3. 创建 Dashboard 展示采集的指标。

5. 性能测试

A. 测试抓取能力

1. 使用 `prometheus-metrics-generator` 生成指标：

```
prometheus-metrics-generator -remoteWrite.url=http://127.0.0.1:8480/api/v1/write -series=100000 -interval=1s
```

2. 检查 `\vmagent`` 的抓取和写入性能：

```
curl http://127.0.0.1:8429/metrics | grep "vm_promscrape_scrapes_total"
```

B. 测试并发写入

1. 使用多个并发连接写入数据：

```
seq 1 100 | xargs -n1 -P10 -I{} curl -X POST "http://127.0.0.1:8480/api/v1/write" -d @test_data.snappy
```

2. 检查 `\vminsert`` 的吞吐量：

```
curl http://127.0.0.1:8480/metrics | grep "vm_rows_inserted_total"
```

6. 高可用性测试

A. 模拟节点故障

1. 停止一个 `\vmstorage`` 节点：

```
systemctl stop vmstorage
```

2. 验证是否能从其他存储节点查询数据：

```
curl "http://127.0.0.1:8481/api/v1/query?query=rate(node_cpu_seconds_total[1m])"
```

B. 数据冗余测试



1. 同时配置两个 `remoteWrite` 地址（主集群和备份集群）。
2. 检查两个集群的存储状态：

```
curl http://127.0.0.1:8482/metrics | grep "vm_rows_inserted_total"
curl http://backup-vmstorage:8482/metrics | grep "vm_rows_inserted_total"
```

7. 监控与日志分析

A. 监控指标

从以下接口获取监控指标：

- `vmagent`: `http://127.0.0.1:8429/metrics`
- `vminsert`: `http://127.0.0.1:8480/metrics`
- `vmstorage`: `http://127.0.0.1:8482/metrics`
- `vmselect`: `http://127.0.0.1:8481/metrics`

B. 常用监控项

- `vm_promscrape_scrapes_total`: 成功抓取的次数。
- `vm_rows_inserted_total`: 成功写入的样本数。
- `vm_rows_queried_total`: 查询的样本数。

C. 日志分析

检查 `vmagent` 和集群的日志：

```
journalctl -u vmagent
journalctl -u vmstorage
```

8. 总结

通过以上流程，可以验证 **vmagent** 是否能稳定高效地抓取数据并写入 VictoriaMetrics 多节点集群，并通过性能测试评估其高并发抓取和写入能力。如果需要进一步优化或解决问题，欢迎继续讨论！



詳細此多節點怎麼跟grafana測試



以下是详细的 **VictoriaMetrics 多节点集群模式** 与 **Grafana** 配合测试的方法。通过 Grafana 配置 VictoriaMetrics 为数据源，创建仪表板，并进行性能测试。



1. 测试目标

1. 验证 Grafana 能否正确查询 VictoriaMetrics 多节点集群的数据。
2. 测试 Grafana 查询性能，评估 VictoriaMetrics 的响应速度和稳定性。
3. 创建仪表板以可视化数据。

2. 环境准备

A. 确保 VictoriaMetrics 多节点部署正常

确认以下组件已正常运行：

- `vmstorage` (数据存储节点)。`
- `vminsert` (写入节点)。`
- `vmselect` (查询节点)。`

检查服务状态：

```
curl http://127.0.0.1:8481/metrics # vmselect
```

B. 安装 Grafana

如果尚未安装 Grafana，可以参考以下命令：

使用 Docker 安装：

```
docker run -d -p 3000:3000 --name=grafana grafana/grafana:latest
```

使用系统包管理工具安装（例如 Ubuntu）：

```
sudo apt-get install -y grafana
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

Grafana 默认运行在 `http://<server-ip>:3000`。`

3. 在 Grafana 配置 VictoriaMetrics 数据源

A. 添加数据源

1. 登录 Grafana Web 界面：
 - 默认用户名：`admin`。`
 - 默认密码：`admin`（首次登录需要修改密码）。`
2. 进入 **Configuration > Data Sources**。
3. 点击 **Add data source**，选择 **Prometheus** 数据源类型。

B. 配置数据源



在数据源配置页面，设置以下参数：

- **URL**：VictoriaMetrics 的 `vmselect` 地址。
 - 示例：`http://127.0.0.1:8481`
- **HTTP Method**：选择 `GET` 或 `POST`（通常为 `GET`）。
- **Custom Headers**：如果需要身份验证，可以添加自定义请求头。
- **Save & Test**：点击按钮，验证连接是否成功。

成功连接后，显示绿色提示 **Data source is working**。

4. 创建仪表板

A. 创建新仪表板

1. 在 Grafana 页面顶部选择 **Dashboards > New Dashboard**。
2. 点击 **Add a new panel**。

B. 配置查询

在查询部分，填写以下 PromQL 查询示例：

1. **单个指标查询**：

```
node_cpu_seconds_total{job="node-exporter"}
```

2. **聚合查询**：

```
sum(rate(node_cpu_seconds_total[5m])) by (instance)
```

3. **复杂查询**：

```
avg(rate(node_network_receive_bytes_total[1m])) by (instance)
```

4. **全局查询**：

```
up
```

C. 配置显示样式

1. 配置图表类型：选择 **Graph**、**Gauge** 或其他类型。
2. 设置时间范围：
 - 默认时间范围：`Last 5 minutes`。
 - 数据刷新频率：如 `5s` 或 `1m`。
3. 点击 **Apply** 保存面板配置。



5. 验证测试结果

A. 验证数据展示

1. 确保面板能正确显示数据，图表无错误提示。
2. 查询数据与预期是否一致。

B. 测试查询性能

1. 在仪表板右上角选择 **Inspect > Query**，查看每次查询的响应时间。
2. 查询耗时应小于 1 秒（常规查询）。

C. 验证大范围查询

1. 查询历史时间范围（如 7 天、30 天）：

```
rate(node_cpu_seconds_total[1h])
```

2. 检查 VictoriaMetrics 是否能快速响应，Grafana 是否正常绘制。

6. 性能测试

A. 模拟高并发查询

1. 使用 `wrk` 工具对 Grafana 数据源进行压测：

```
wrk -t4 -c100 -d30s "http://127.0.0.1:3000/datasource_id/prometheus/api/v1/query?query=rate(node_cpu_seconds_total[5m])"
```

参数说明：

- `-t4`：4 个线程。
 - `-c100`：100 并发连接。
 - `-d30s`：持续测试 30 秒。
2. 检查 VictoriaMetrics 的查询性能：

```
curl http://127.0.0.1:8481/metrics | grep "vm_rows_queried_total"
```

B. 测试资源占用

通过 `htop` 或 `top` 监控以下组件的资源使用情况：

- Grafana（CPU、内存）。
- VictoriaMetrics（特别是 `vmselect`）。

7. 高可用与扩展性测试

A. 配置负载均衡

1. 如果 `vmselect` 有多个副本，可以在 Grafana 数据源 URL 中添加负载均衡地址（如使用 Nginx 或 HAProxy）。



2. 配置 Nginx 负载均衡：

```
upstream vmselect {
    server vmselect1:8481;
    server vmselect2:8481;
}

server {
    listen 8080;

    location / {
        proxy_pass http://vmselect;
    }
}
```

3. 更新 Grafana 数据源：

```
URL: http://127.0.0.1:8080
```

B. 模拟 `vmselect` 故障

1. 停止一个 `vmselect` 节点：

```
systemctl stop vmselect
```

2. 验证 Grafana 是否仍能通过其他节点查询数据。

8. 日志分析与调优

A. Grafana 查询日志

1. 检查查询日志：

- Grafana 日志文件位置（取决于安装方式）：
 - 默认：`/var/log/grafana/grafana.log`
 - Docker：`docker logs grafana`

2. 重点查看错误查询和长时间查询。

B. VictoriaMetrics 日志

1. 检查 `vmselect` 的查询日志：

```
journalctl -u vmselect
```

2. 查看性能指标：

- `vm_query_duration_seconds`：查询耗时。
- `vm_query_rows_queried_total`：查询的行数。

9. 总结



通过以上步骤，可以验证 VictoriaMetrics 多节点集群和 Grafana 的集成是否正常。测试包括数据源配置、仪表板创建、查询性能、扩展性和高可用性等。

如果需要更复杂的仪表板或定制查询，请进一步说明需求！



詳細比較有無使用k8s來部屬多節點的差異



以下是 **使用 Kubernetes (K8s)** 和**不使用 Kubernetes** 部署 VictoriaMetrics 多节点集群的详细比较，包括各方面的优劣势分析，以及适用场景：

1. 部署方式概述

部署方式	概述
使用 Kubernetes	在 K8s 集群中运行多个 VictoriaMetrics 组件，自动管理容器、资源分配等。
不使用 Kubernetes	手动在裸机或虚拟机上运行 VictoriaMetrics，通过手动配置和管理节点。

2. 比较维度

A. 安装与配置

维度	使用 Kubernetes	不使用 Kubernetes
部署复杂性	使用 Helm Chart 或 YAML 文件，一键式部署，自动处理依赖和资源分配。	手动下载、解压、配置和启动各组件，需逐个管理端口和参数。
资源配置	可通过 K8s 配置文件统一管理资源（如 CPU、内存、存储）。	需单独为每个节点配置资源，难以批量管理。
初始学习曲线	较高，需要熟悉 Kubernetes 基础概念，如 Pod、Service、Deployment。	较低，只需基本的 Linux 操作和服务管理知识。

B. 节点扩展

维度	使用 Kubernetes	不使用 Kubernetes
水平扩展	轻松扩展，通过调整副本数（`replicas`）扩展 `vmstorage`、`vminsert`、`vmselect` 节点。	手动启动新节点并更新现有组件的配置文件（如 `vminsert` 的 `storageNode`）。
自动扩展	支持自动扩展（如 HPA，基于 CPU/内存负载自动增加副本）。	无法自动扩展，需人工判断负载并添加节点。
灵活性	可以快速适应负载变化，资源分配灵活，适合大规模集群。	对负载变化响应较慢，扩展效率低，适合固定规模的小集群。



C. 高可用性

维度	使用 Kubernetes	不使用 Kubernetes
组件故障恢复	Pod 异常时会自动重后，Deployment 会维持副本数一致性。	手动检查并重后服务，恢复时间较长。
负载均衡	K8s 自动提供内部负载均衡（如 Service），对查询和写入节点均衡流量。	需手动配置 Nginx 或 HAProxy 实现负载均衡。
故障容错	多节点容错，故障节点被隔离，保证整体服务可用性。	故障节点需手动修复或切换，期间可能影响可用性。

D. 运维管理

维度	使用 Kubernetes	不使用 Kubernetes
服务管理	使用 `kubect1` 管理组件（如滚动升级、回滚、重后）。	使用 `systemctl` 或手动运行命令管理服务，复杂度较高。
监控与日志	内置监控工具（如 Prometheus Operator、K8s Dashboard），集中管理。	分散管理日志和监控（如单独配置 Prometheus 监控每个节点）。
故障排查	支持 `kubect1 logs` 查看组件日志，快速定位问题。	手动 SSH 登录每个节点检查日志和状态，效率较低。

E. 性能与资源开销

维度	使用 Kubernetes	不使用 Kubernetes
系统资源消耗	K8s 本身需要额外的资源（如控制平面和网络插件），对于小型部署可能显得浪费。	资源消耗仅与 VictoriaMetrics 本身相关，更加高效。
性能稳定性	更加稳定，K8s 提供自动恢复和健康检查机制，减少意外宕机影响。	稳定性取决于手动配置和运维能力，容易因疏漏导致性能问题。

F. 成本与适用场景

维度	使用 Kubernetes	不使用 Kubernetes
运维成本	高效但需要初期学习成本，适合已使用 K8s 的团队。	运维成本较低，但大规模集群运维复杂度高，效率较低。
适用场景	- 大规模部署（高并发、百万级指标场景）。 - 动态负载变化的场景。	- 小规模集群或固定部署。 - 资源有限的小型环境（如个人实验）。

3. 综合优劣势分析

使用 Kubernetes

优点：

- 部署和扩展更加高效，适合大规模、动态变化的集群。





- 高可用性强，故障自动恢复，减少宕机时间。
- 集成监控和日志工具，方便集中化管理。
- 支持自动化操作（滚动升级、扩展、回滚）。

缺点：

- 初始学习成本高，需掌握 K8s 的概念和操作。
- K8s 集群本身消耗额外的硬件资源。

不使用 Kubernetes

优点：

- 部署简单，适合小规模环境。
- 资源开销小，不需要额外运行 Kubernetes 控制平面。
- 更适合不熟悉 Kubernetes 的团队。

缺点：

- 缺乏自动化管理，手动操作多，运维复杂度高。
- 扩展效率低，难以快速响应动态负载变化。
- 容错性弱，需人工处理节点故障和流量均衡。

4. 选择建议

场景	推荐部署方式
小型集群（1~3 个节点），固定负载	不使用 Kubernetes
大规模集群（10+ 节点），动态负载	使用 Kubernetes
已经有 Kubernetes 集群的环境	使用 Kubernetes
资源受限，难以支持 Kubernetes 集群的环境	不使用 Kubernetes
对高可用性和快速扩展有强需求的生产环境	使用 Kubernetes
学习或实验目的	不使用 Kubernetes

通过以上对比，您可以根据自身场景和需求选择合适的部署方式。如需具体操作指导或调整优化，请随时联系！