



# 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"，詳細一步步教我怎麼做，能比較出與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 天以上的时间序列数据，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 `kubect1 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=1
```

### 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！