大规模集群部署问题

prometheus本身只支持单机部署，没有自带集群模式，所以在目前大规模集群监控主要通过prometheus联邦机制或者做监控指标服务拆分方式去实现并且最难以接受的是prometheus对于监控历史数据的存储问题，在本地不能存储过久的监控数据，只能通过远端存储接口，存储到支持prometheus远端存储接口的数据库中。（因为Prometheus将采集的样本放到内存中，默认每隔2小时将数据压缩成一个block，持久化到硬盘中，样本的数量越多，Prometheus占用的内存就越高，会造成OOM的情况，虽然可以优化拉取时间和和加快落盘时间 但是只是缓解，不能解决根本问题）

然后远程存储是通过将WAL中的数据缓存到多个内存队列（shards）中，然后写到远程存储设备，其直接与WAL打交道。

但是这带来的问题就是引入新的组件会增加对应的运维工作量。

prometheus高可用问题和扩展问题

prometheus官方高可用的方式是通过部署多个prometheus实例采集同一个target，前端通过LB设备做为统一入口，这带来的问题就是，两个prometheus实例内存储的数据会存在差异，特别是当其中一个prometheus宕机后，另外一个prometheus接管服务，此时宕机的prometheus就会丢失宕机期间的监控数据，当LB的请求转发过来会出现数据不一致情况。

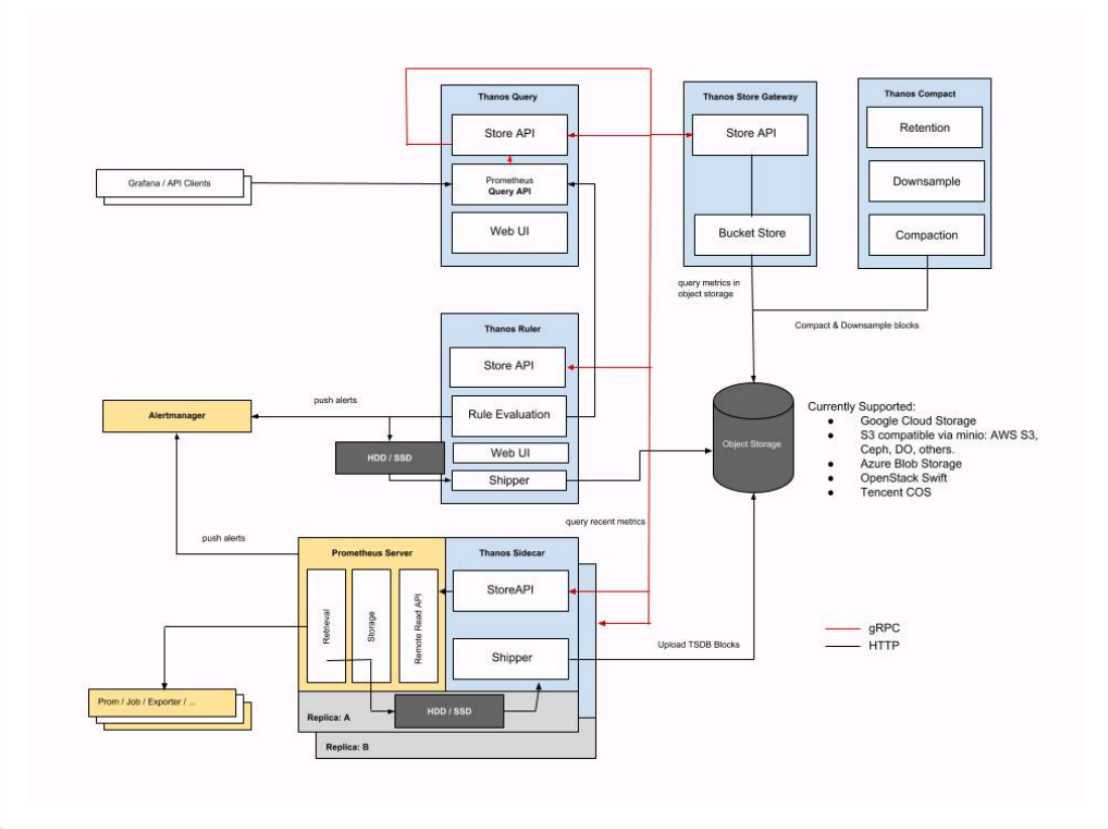
Prometheus将采集的样本放到内存中，默认每隔2小时将数据压缩成一个block，持久化到硬盘中，样本的数量越多，Prometheus占用的内存就越高，会造成OOM的情况，虽然可以优化拉取时间和和加快落盘时间 但是只是缓解，不能解决根本问题

社区出现了一个Prometheus的集群解决方案：Thanos，它提供了全局查询视图，可以从多台Prometheus查询和聚合数据，因为所有这些数据均可以从单个端点获取。

Thanos能解决什么问题?

Thanos能够解决上述问题，thanos能够将多个prometheus实例的数据进行聚合去重，来支持prometheus横向扩展和提高prometheus的高可用性，同时也支持将历史监控数据存储到对象存储中，提供监控数据的可靠性，降低运维难度。

Thanos 主要由如下几个特定功能的组件组成：



thanos-sidecar（接口容器）： 通过Prometheus附加件与Prometheus进行连接，通过http方式在Prometheus的remote-read API基础之上实现了storeAPI接口，query组件可以直接从此接口读取监控数据，以供实时查询，并且还支持将Prometheus数据上传到对应的对象存储,以供长期保存

thanos-query（查询网关)： 通过与thanos-sidecar组件的store-api的grpc接口抓取监控数据，并对监控数据进行聚合去重处理。

thanos-storage-gateway (存储网关)： 对接后端对象存储，当需要查询对象存储中的历史监控数据时，将云存储中的数据内容暴露出来,供给query相连查看。

thanos-compact（压缩器）： 对存储在对象存储中的监控数据多个较小的块连续合并为较大的块。这显着减少了存储桶中的总存储大小。提高查询效率。

thanos-rules： 对监控数据进行告警，通知altermanager，并且可以预先计算经常需要或计算量大的表达式，并将其结果保存为一组新的时间序列据提供给query查询和对象存储进行存储。

- 接收器（Receiver）：从 Prometheus 的 remote-write WAL（Prometheus 远程预写式日志）获取数据，暴露出去或者上传到云存储
- 规则组件（Ruler）：针对监控数据进行评估和报警

- Bucket：主要用于展示对象存储中历史数据的存储情况，查看每个指标源中数据块的压缩级别，解析度，存储时段和时间长度等信息

有了Thanos之后，Prometheus的水平扩展就能变得更加简单，不仅如此，Thanos还提供了可靠的数据存储方案，可以监听和备份prometheus本地数据到远程存储。

操作步奏

首先我们看到构架图后端最核心的就是一个对象存储，这里我们采用的是自建的对象存储 Minio

安装 Minio

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，

例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

要安装 Minio 非常容易的，同样我们这里将 Minio 安装到 Kubernetes 集群中，可以直接参考官方文档 使用Kubernetes部署MinIO，

在 Kubernetes 集群下面可以部署独立、分布式或共享几种模式，可以根据实际情况部署，我们这里为了简单直接部署独立模式。

直接使用 Deployment 来管理 Minio 的服务：（minio-deploy.yaml）

```
kubectl create ns minio

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-pvc
spec:
  storageClassName: nfs-storageclass
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi

---
apiVersion: v1
kind: Service
metadata:
  name: minio
spec:
  ports:
  - port: 9000
    targetPort: 9000
    protocol: TCP
  selector:
    app: minio

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: minio
spec:
  selector:
    matchLabels:
      app: minio
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: minio
    spec:
      volumes:
      - name: data
        persistentVolumeClaim:
```

```
          claimName: minio-pvc
      containers:
      - name: minio
        volumeMounts:
        - name: data
          mountPath: "/data"
        image: minio/minio:RELEASE.2020-11-06T23-17-07Z
        args:
        - server
        - /data
        env:
        - name: MINIO_ACCESS_KEY
          value: "minio"
        - name: MINIO_SECRET_KEY
          value: "minio123"
        ports:
        - containerPort: 9000
        resources:
          requests:
            cpu: 100m
            memory: 512Mi
          limits:
            cpu: 1000m
            memory: 2048Mi
        readinessProbe:
          httpGet:
            path: /minio/health/ready
            port: 9000
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 3
        livenessProbe:
          httpGet:
            path: /minio/health/live
            port: 9000
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
```

创建一个对象存储配置文件：（**09thanos-storage-minio.yaml**）

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-objectstorage
  namespace: monitoring
type: Opaque
data: {}
stringData:
  thanos.yaml: |-
    type: s3
    config:
      bucket: prometheus-thanos
      endpoint: minio.minio.svc.cluster.local:9000
      access_key: minio
      secret_key: minio123
      insecure: true
      signature_version2: false
```

Sidecar 组件

由于prometheus-operator 支持thanos扩展，我们直接在prometheus-prometheus.yaml最后添加 thanos 配置

```
  thanos:  #  添加 thanos 配置
    objectStorageConfig:
      key: thanos.yaml
      name: thanos-objectstorage  # 对象存储对应的 secret 资源对象

  #podMetadata标签增加thanos-store-api: "true"
  podMetadata:
    labels:
      thanos-store-api: "true"
```

Querier 组件

通过与thanos-sidecar组件的store-api的grpc接口抓取监控数据

```
apiVersion: v1
kind: Service
metadata:
  name: thanos-querier
  namespace: monitoring
  labels:
    app: thanos-querier
spec:
  ports:
  - port: 9090
    protocol: TCP
    targetPort: http
    nodePort: 60000
    name: http
  selector:
    app: thanos-querier
  type: NodePort

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thanos-querier
  namespace: monitoring
  labels:
    app: thanos-querier
spec:
  selector:
    matchLabels:
      app: thanos-querier
  template:
    metadata:
      labels:
        app: thanos-querier
    spec:
      containers:
      - name: thanos
        image: thanosio/thanos:v0.11.0
        args:
        - query
        - --log.level=debug
        - --query.replica-label=prometheus_replica
        # Discover local store APIs using DNS SRV.
```

```
        - --store=dnssrv+thanos-store-gateway:10901
        ports:
        - name: http
          containerPort: 10902
        - name: grpc
          containerPort: 10901
        resources:
          requests:
            memory: "2Gi"
            cpu: 500m
          limits:
            memory: "4Gi"
            cpu: "2"
        livenessProbe:
          httpGet:
            path: /-/healthy
            port: http
          initialDelaySeconds: 10
        readinessProbe:
          httpGet:
            path: /-/healthy
            port: http
          initialDelaySeconds: 15
```

Compactor 组件

Thanos 的 Compactor 组件，用来将对象存储中的数据进行压缩和下采样。

Compactor 组件的部署和 Store 非常类似，指定对象存储的配置文件即可，如下所示的资源清单文件：（compactor.yaml）

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: thanos-compactor
  namespace: monitoring
  labels:
    app: thanos-compactor
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: thanos-compactor
  serviceName: thanos-compactor
  template:
    metadata:
      labels:
        app: thanos-compactor
    spec:
      containers:
      - name: thanos
        image: thanosio/thanos:v0.11.0
        args:
        - "compact"
        - "--log.level=debug"
        - "--data-dir=/data"
        - "--objstore.config-file=/etc/secret/thanos.yaml"
        - "--wait"
        ports:
        - name: http
          containerPort: 10902
        livenessProbe:
          httpGet:
            port: 10902
            path: /-/healthy
          initialDelaySeconds: 10
        readinessProbe:
          httpGet:
            port: 10902
            path: /-/ready
          initialDelaySeconds: 15
        volumeMounts:
        - name: object-storage-config
          mountPath: /etc/secret
          readOnly: false
      volumes:
      - name: object-storage-config   # 挂载对象存储配置
        secret:
          secretName: thanos-objectstorage
```

安装 Thanos Store

Prometheus和thanos-store-gateway的有状态集被标记为thanos-store-api：“true”这样网页上面就能看到冷热数据页面了

```
apiVersion: v1
kind: Service
metadata:
  name: thanos-store-gateway
  namespace: monitoring
spec:
  type: ClusterIP
  clusterIP: None
  ports:
  - name: grpc
    port: 10901
    targetPort: grpc
  selector:
    thanos-store-api: "true"

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: thanos-store-gateway
  namespace: monitoring
  labels:
    app: thanos-store-gateway
spec:
  replicas: 1
  selector:
    matchLabels:
      app: thanos-store-gateway
  serviceName: thanos-store-gateway
  template:
    metadata:
      labels:
        app: thanos-store-gateway
        thanos-store-api: "true"
    spec:
      containers:
        - name: thanos
          image: thanosio/thanos:v0.11.0
          args:
          - "store"
```

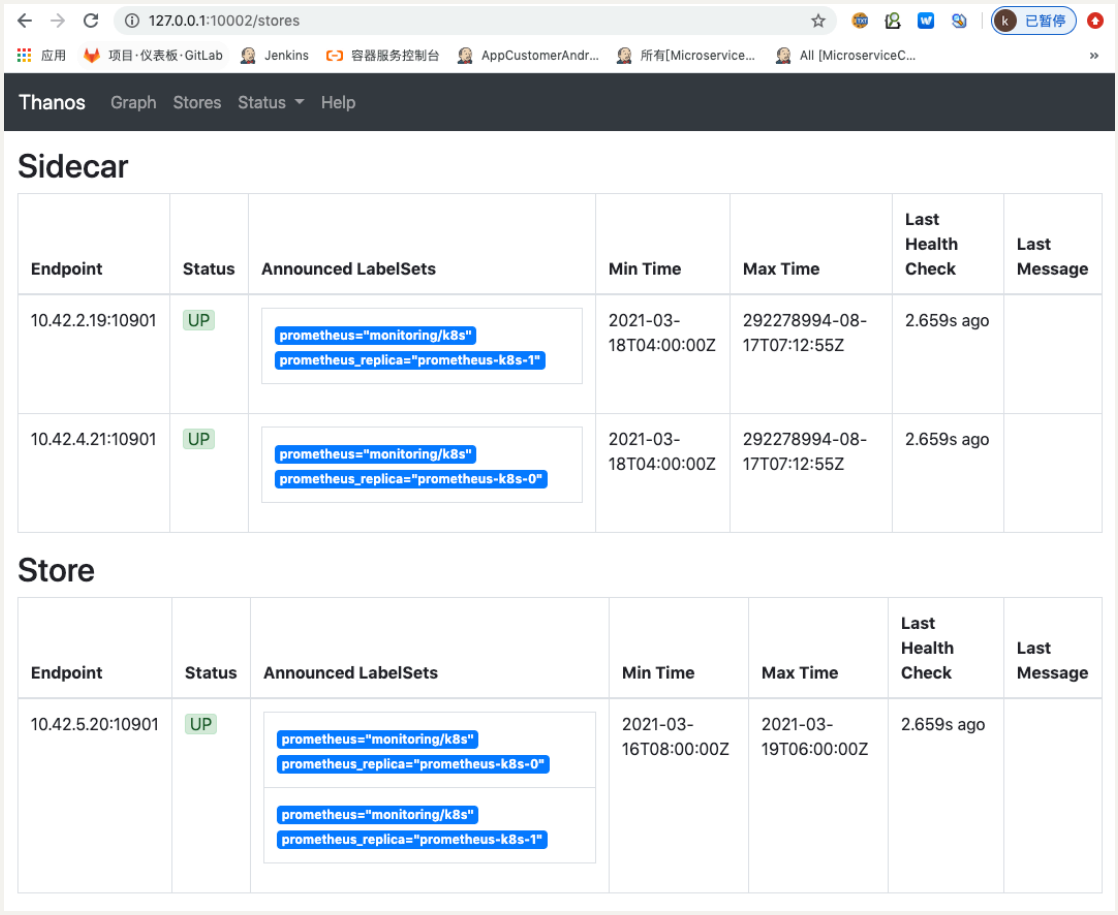
```
          - "--log.level=debug"
          - "--data-dir=/data"
          - "--objstore.config-file=/etc/secret/thanos.yaml"
          - "--index-cache-size=500MB"
          - "--chunk-pool-size=500MB"
          ports:
          - name: http
            containerPort: 10902
          - name: grpc
            containerPort: 10901
          livenessProbe:
            httpGet:
              port: 10902
              path: /-/healthy
          readinessProbe:
            httpGet:
              port: 10902
              path: /-/ready
          resources:
            requests:
              memory: "2Gi"
              cpu: 500m
            limits:
              memory: "4Gi"
              cpu: "2"
          volumeMounts:
            - name: object-storage-config
              mountPath: /etc/secret
              readOnly: false
      volumes:
        - name: object-storage-config
          secret:
            secretName: thanos-objectstorage
```

之后本地代理thanos-querier，就能看到冷热数据了

kubectl port-forward services/thanos-querier 10002:9090 -n monitoring

Sidecar是热数据，连接着Prometheus数据库

Store是冷数据，连接着Thanos Store数据库



然后更改grafana-dashboardDatasources.yaml （base64解码 然后更改后编码）更改url为http://thanos-querier:9090

让数据的获取都从thanos-querier走

```
{
    "apiVersion": 1,
    "datasources": [
        {
            "access": "proxy",
            "editable": false,
            "name": "prometheus",
            "orgId": 1,
            "type": "prometheus",
            "url": "http://thanos-querier:9090",
            "version": 1
        }
    ]
}
```

Prometheus章节完