云计算技术 · 期末实验大作业答辩

K8s可观测性最小闭环系统建设

选题编号: 09 / 监控+日志+告警最小闭环(挑战)

小组成员

肖思梦、严羽、张馨尹、张婉婷

答辩日期

2026-01-05

目录

01

项目背景与目标

项目背景与目标

问题背景

监控分散

K8s集群运维中监控数据分散在不同平台,缺乏统一视图,难以快速定位问题

日志难查

日志分散在各Pod和节点,没有集中化存储和检索能力,排查效率低下

告警缺失

缺少自动化告警机制,无法及时发现和处理集群异常,影响系统稳定性

项目目标

构建一体化闭环

打造"监控-日志-告警"三位一体的一体化可观测性闭环系统

指标可视化

通过Grafana实现监控指标的可视化展示,提供直观的集群状态视图

日志可追溯

基于Loki实现日志的集中化采集、存储和检索,支持快速问题溯源

告警可触发

配置PrometheusRule告警规则,实现异常的自动检测和实时告警通知

不做范围

多集群监控

专注于单集群环境,不涉及多集群联邦监控的复杂场景

自定义插件

使用标准组件,不开发自定义Grafana插件

多渠道告警

告警触发后仅在Prometheus平台展示,不集成邮件/钉钉/微信等渠道

聚焦核心功能

02

关键概念与原理

关键概念与原理

概念1: 指标监控 Prometheus

Pull模式采集

Prometheus主动拉取各Target指标数据,支持HTTP协议定期采集

时序数据存储

采用高效时序数据库TSDB,按时间戳和标签存储监控指标数据

PromQL查询

提供强大查询语言,支持复杂聚合、计算和可视化分析

概念2: 日志集中化 Loki

标签索引

只对日志标签建立索引,原始日志内容压缩存储,大幅降低存储成本

原始存储

日志原文按块存储在对象存储中,支持高效压缩和批量读取

低延迟查询

基于标签索引快速定位日志块,支持LogQL语法高效查询过滤

概念3: 告警规则 PrometheusRule

基于阈值检测

通过PromQL表达式定义阈值条件,持续评估指标是否触发告警

异常检测机制

支持持续时间(for)设置,避免瞬时波动误报,确保告警准确性

主动告警

告警触发后状态从Pending转为Firing,可在Grafana界面实时查看

03

系统架构与拓扑图

系统架构与拓扑图

四层架构拓扑图

功能应用层 — Layer 4

- 告警管理
- 日志查询
- 可视化

核心服务层 — Layer 3

- Prometheus v0.14
- Loki v3.6.3
- Grafana v9.5.2

数据采集层 — Layer 2

- node-exporter
- kube-state-metrics
- Alloy v1.5.1

基础设施层 — Layer 1

- Kubernetes
- Docker
- OpenEBS

层与层之间通过标准接口通信,实现解耦和可扩展性

数据流说明

1 指标流

node-exporter/kube-state-metrics → Prometheus → Grafana

2 日志流

应用Pod → Alloy → Loki → Grafana

3 告警流

Prometheus → Pending/Firing → Grafana/prometheus页面

架构特点

- 模块化设计
- 高可扩展
- 高可用
- 标准化接口

04

核心组件职责表

核心组件职责表

组件名称	核心职责	版本	所属层级
Prometheus	时序指标存储、告警规则配置与触发判断，采集间隔≤30s	v0.14	核心服务层
Loki	日志集中存储与标签索引，支持按 Pod、命名空间筛选，日志延迟≤1min	v3.6.3	核心服务层
Grafana	指标可视化仪表盘展示、Loki 日志查询集成，提供统一运维入口	v9.5.2	核心服务层
Alloy	全集群 Pod 日志采集与转发，支持按标签过滤	v1.5.1	数据采集层
OpenEBS	提供本地存储卷，保障 Loki 日志数据持久化存储	-	基础设施层

各组件职责清晰,协同构建完整的可观测性解决方案

共5个核心组件,覆盖监控、日志、告警全链路

05

实现步骤

实现步骤1 - 环境准备与Docker优化

Docker镜像加速配置

问题:国内镜像拉取缓慢

由于国内网络环境,Docker Hub镜像拉取速度慢,影响部署效率

解决方案:

配置国内镜像加速器提供的Docker镜像加速服务

配置示例:

```
yanyu@master:~/kube-prometheus$ sudo tee /etc/docker/daemon.json <<-'EOF'
> {
> "registry-mirrors": ["https://docker.m.daocloud.io/",
>     "https://dockerhub.icu",
>     "https://docker.chenby.cn",
>     "https://docker.1panel.live",
>     "https://docker.awsl9527.cn",
>     "https://docker.anyhub.us.kg",
>     "https://dhub.kubesre.xyz",
>     "https://docker.13140521.xyz"
> ]
> }
> EOF
```

网络策略与访问配置

NodePort服务暴露

将Service类型改为NodePort，通过节点端口实现外部访问Grafana

```
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: web
    port: 9090
    protocol: TCP
    targetPort: web
  - name: reloader-web
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
  type: NodePort
status:
  loadBalancer: {}
```

网络策略调整

删除monitoring命名空间下的所有NetworkPolicy，确保Web视图可访问

```
yanyu@master:~/kube-prometheus/manifests$ kubectl  -n monitoring  delete  networkpolicies grafana
networkpolicy.networking.k8s.io "grafana" deleted
yanyu@master:~/kube-prometheus/manifests$ kubectl  -n monitoring  delete  networkpolicies prometheus-k8s
networkpolicy.networking.k8s.io "prometheus-k8s" deleted
```

环境准备检查清单

- Docker加速配置
- K8s集群就绪
- 节点端口开放
- 网络策略配置

实现步骤2 - Prometheus+Grafana部署

kube-prometheus部署

CRD + ServiceMonitor模式

采用Operator模式部署,通过CRD定义Prometheus、Alertmanager等资源,使用ServiceMonitor自动发现监控目标

```
yanyu@master:~$ git clone -b release-0.14 https://github.com/prometheus-operator/kube-prometheus.git
正克隆到 'kube-prometheus'...
remote: Enumerating objects: 22260, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 22260 (delta 2), reused 2 (delta 1), pack-reused 22252 (from 1)
接收对象中: 100% (22260/22260), 14.60 MiB | 14.73 MiB/s, 完成.
处理 delta 中: 100% (15590/15590), 完成.
```

镜像拉取优化

替换为国内镜像源,加速镜像拉取速度

```
    app.kubernetes.io/version: 0.12.0
spec:
  automountServiceAccountToken: true
  containers:
  - args:
    - --cert-dir=/var/run/serving-cert
    - --config=/etc/adapter/config.yaml
    - --metrics-relist-interval=1m
    - --prometheus-url=http://prometheus-k8s.monitoring.svc:9090/
    - --secure-port=6443
    - --tls-cipher-suites=TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
S_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AE
_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_CBC_SHA,TL
_WITH_AES_256_CBC_SHA
    image: swr.cn-north-4.myhuaweicloud.com/ddn-k8s/registry.k8s.io/prometheus-adapter/prometheus-adapter:v0.12.0
    livenessProbe:
      failureThreshold: 5
      httpGet:
        path: /livez
```

Service改为NodePort

外部访问配置

修改grafana-service.yaml,将类型从ClusterIP改为NodePort,通过节点端口暴露服务

```
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: web
    port: 9090
    protocol: TCP
    targetPort: web
  - name: reloader-web
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
  type: NodePort
status:
  loadBalancer: {}
```

```
yanyu@master:~/kube-prometheus/manifests$ kubectl  -n monitoring  get svc
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                         AGE
alertmanager-main       ClusterIP   10.105.143.113   <none>        9093/TCP,8080/TCP               25m
alertmanager-operated   ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP      24m
blackbox-exporter       ClusterIP   10.97.139.169    <none>        9115/TCP,19115/TCP              25m
grafana                 NodePort    10.100.148.177   <none>        3000:30999/TCP                  25m
kube-state-metrics      ClusterIP   None             <none>        8443/TCP,9443/TCP               25m
node-exporter           ClusterIP   None             <none>        9100/TCP                        25m
prometheus-adapter      ClusterIP   10.105.8.53      <none>        443/TCP                         25m
prometheus-k8s          NodePort    10.111.198.85    <none>        9090:32703/TCP,8080:30808/TCP   25m
prometheus-operated     ClusterIP   None             <none>        9090/TCP                        24m
prometheus-operator     ClusterIP   None             <none>        8443/TCP                        25m
yanyu@master:~/kube-prometheus/manifests$
```

默认账号密码

用户名	密码
admin	admin

⚠ 首次登录需修改密码

实现步骤3 - 自定义告警规则设计

1 NodeNotReady

告警描述

节点异常,当K8s节点状态为NotReady时触发告警

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    prometheus: k8s
    role: alert-rules
  name: prometheus-k8s-rules
  namespace: monitoring
spec:
  groups:
  - name: NodeExport.rules
    rules:
    #主机状态
    - alert: NodeNotReady
      annotations:
        description: 'Node:{{ $labels.instance }},主机状态异常'
        summary: 主机状态异常
      expr: |
        up{job="node-exporter"} == 0
      for: 3m
```

for: 3m

2 EasyMemoryAlert

告警描述

内存超限(测试用),当节点内存使用率超过阈值时触发告警

```
^Cyanyu@master:~/kube-prometheus/manifests$ cat > easy-alert-fixed-final.yaml << 'EOF'
> apiVersion: monitoring.coreos.com/v1
> kind: PrometheusRule
> metadata:
>   name: easy-memory-alert
>   namespace: monitoring
> spec:
>   groups:
>   - name: easy.rules
>     rules:
>     - alert: EasyMemoryAlert
>       annotations:
>         description: '测试告警 - 内存使用超过 1字节'
>         summary: '内存测试告警（最终版）'
>       expr: container_memory_working_set_bytes > 1  # 去掉 container 筛选
>       for: 10s
> EOF
```

for: 10s

3 PodRestartFrequent

告警描述

Pod频繁重启,当Pod重启次数超过阈值时触发告警

```
^Cyanyu@master:~/kube-prometheus/manifests$ cat > pod-restart-alert.yaml << 'EOF'
> apiVersion: monitoring.coreos.com/v1
> kind: PrometheusRule
> metadata:
>   name: pod-restart-alert
>   namespace: monitoring
> spec:
>   groups:
>   - name: pod.rules
>     rules:
>     - alert: PodRestartFrequent
>       annotations:
>         description: 'Pod {{ $labels.pod }} 在10分钟内重启了 {{ $value }} 次'
>         summary: 'Pod频繁重启'
>       expr: increase(kube_pod_container_status_restarts_total[10m]) > 3
>       for: 1m
>       labels:
>         severity: warning
> EOF
```

for: 1m

规则配置文件

通过kubectl apply -f 部署
告警规则自动加载生效

实现步骤4 - Loki日志系统部署

OpenEBS存储部署

持久化支持

部署OpenEBS LocalPV存储类,为Loki提供持久化存储支持,确保日志数据在Pod重启后不丢失

```
yanyu@master:~$ vi openebs.yaml
yanyu@master:~$ cat openebs.yaml
# This manifest is autogenerated via `make manifests` command
# Do the modification to the device-driver.yaml in directory deploy/yamls/
# and then run `make manifests` command

# This manifest deploys the OpenEBS Device control plane components,
# with associated CRs & RBAC rules.

# Create the OpenEBS namespace
# This is the default namespace where the Device driver will create all
# its resources. If we want to change it to use a different namespace
# modify this to create the new namespace and also modify the DEVICE_DRIVER_NAMESPACE
# env for the Device Driver's controller and agent deployments.
# please note that this should be changed while initial setup, once Device Driver
# is deployed with a namespace, we should never modify it as old resources will
# not be available under the new namespace and Device Driver looks for all the resources
# in the DEVICE_DRIVER_NAMESPACE namespace passed as an env.

apiVersion: v1
kind: Namespace
metadata:
  name: openebs


##############################################
###########                     ############
###########   DeviceVolume CRD  ############
###########                     ############
##############################################

# DeviceVolume CRD is autogenerated via `make manifests` command.
# Do the modification in the code and run the `make manifests` command
# to generate the CRD definition
```

Alloy日志客户端部署

DaemonSet部署模式

以DaemonSet方式在每个K8s节点部署Alloy Pod,采集该节点上所有容器的日志

离线安装方式

1. 下载Alloy Helm chart
2. 编辑配置文件
3. 安装插件

forward_to: [loki.write.local]

Loki离线安装与配置

离线安装方式

由于网络限制无法在线安装，采用离线安装:

1. 下载Loki Helm chart
2. 上传到机器
3. 编辑配置文件
4. 安装Loki

验证Loki状态

```
kubectl get pods -n loki
```

Grafana集成Loki数据源

数据源添加

在Grafana界面添加Loki数据源,配置Loki服务地址,实现日志查询功能

数据源地址:

```
http://loki-read.loki.svc.cluster.local:3100
```

验证数据源

在Grafana中点击"Save & Test"，验证连接成功

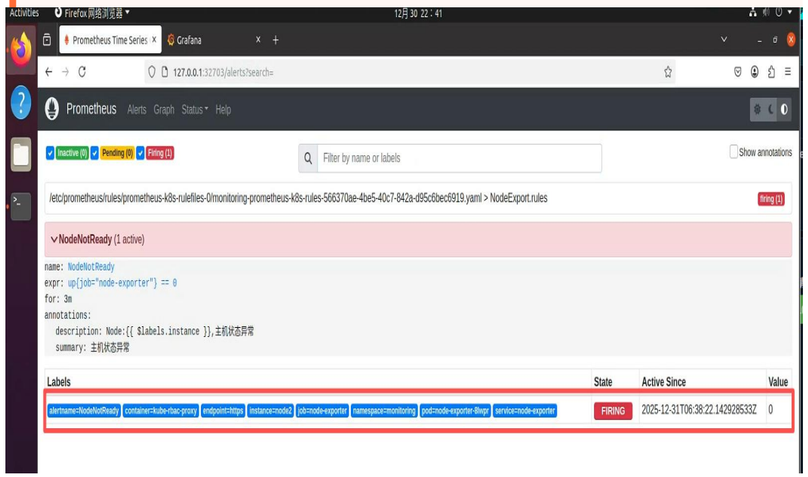
实现步骤5 - 故障模拟与告警验证

节点下线

触发告警

NodeNotReady

模拟K8s节点故障,通过kubectl drain命令将节点下线,验证节点异常告警



Pod频繁重启

触发告警

PodRestartFrequent

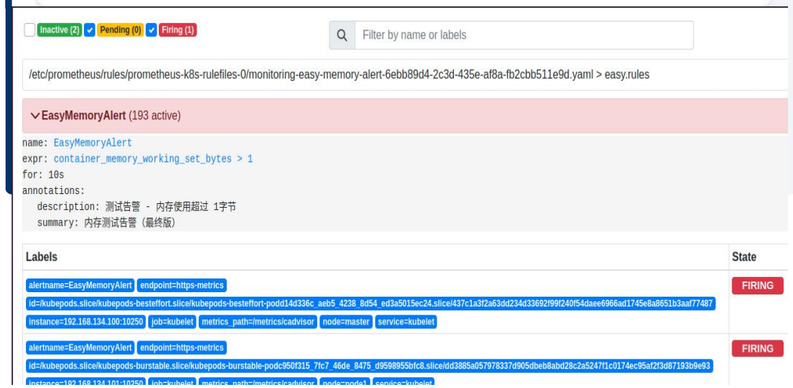
部署一个会频繁崩溃的容器应用,模拟Pod频繁重启场景

内存超限

触发告警

EasyMemoryAlert

部署内存泄漏应用或使用stress工具,模拟内存使用率超过阈值



告警验证成功

三条告警规则均能正常触发

告警延迟: ≤5min

满足实时性要求

实现步骤6 - Grafana扩缩容测试

扩容操作

副本数 1 → 2

执行命令

```
kubectl scale deployment grafana --replicas=2 -n monitoring
```

验证结果

Deployment状态：2/2
Pod状态：Running

调度时间

新Pod在30秒内完成调度，并很快进入Running状态

缩容操作

副本数 2 → 1

执行命令

```
kubectl scale deployment grafana --replicas=1 -n monitoring
```

验证结果

Deployment状态：1/1
Pod终止方式：优雅终止

终止时间

多余Pod在10秒内优雅终止，服务无中断

```
yanyu@master:~/kube-prometheus/manifests$ kubectl get deployment grafana -n monitoring
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
grafana   1/1     1            1           46h
yanyu@master:~/kube-prometheus/manifests$ kubectl scale deployment grafana --replicas=2 -n monitoring
deployment.apps/grafana scaled

yanyu@master:~/kube-prometheus/manifests$ kubectl get pods -n monitoring | grep grafana
grafana-7bd56db9ff-jmgmc              1/1     Running   0              11s
grafana-7bd56db9ff-rpbjb              1/1     Running   4 (85m ago)    42h
yanyu@master:~/kube-prometheus/manifests$ docker images | grep grafana
grafana/alloy                                                                   v1.12.1
                  a7be079a316a   2 weeks ago     496MB
grafana/loki                                                                    3.6.3
                  06c425ea32fa   3 weeks ago     132MB
grafana/loki-canary                                                             3.6.3
                  2764f5ea7576   3 weeks ago     37.4MB
grafana/grafana                                                                 9.5.2
                  6c5313ea00cc   2 years ago     309MB
yanyu@master:~/kube-prometheus/manifests$ kubectl get deployment grafana -n monitoring
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
grafana   2/2     2            2           46h
```

```
yanyu@master:~/kube-prometheus/manifests$ kubectl scale deployment grafana --replicas=1 -n monitoring
deployment.apps/grafana scaled
yanyu@master:~/kube-prometheus/manifests$ sleep 5
yanyu@master:~/kube-prometheus/manifests$ kubectl get pods -n monitoring | grep grafana
grafana-7bd56db9ff-rpbjb              1/1     Running   4 (89m ago)    42h
yanyu@master:~/kube-prometheus/manifests$ kubectl get deployment grafana -n monitoring
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
grafana   1/1     1            1           46h
yanyu@master:~/kube-prometheus/manifests$
```

实现步骤7 - HPA扩缩容测试

创建一个用于 Grafana 的 Horizontal Pod Autoscaler（HPA）配置文件

```
yanyu@master:~/桌面$ cat > grafana-hpa.yaml << 'EOF'
> apiVersion: autoscaling/v2
> kind: HorizontalPodAutoscaler
> metadata:
>   name: grafana-hpa
>   namespace: monitoring
> spec:
>   scaleTargetRef:
>     apiVersion: apps/v1
>     kind: Deployment
>     name: grafana
>   minReplicas: 1
>   maxReplicas: 3
>   metrics:
>   - type: Resource
>     resource:
>       name: cpu
>       target:
>         type: Utilization
>         averageUtilization: 40  # CPU 使用率超过 40% 时扩容
>   - type: Resource
>     resource:
>       name: memory
>       target:
>         type: Utilization
>         averageUtilization: 70  # 内存使用率超过 70% 时扩容
>   behavior:
>     scaleDown:
>       stabilizationWindowSeconds: 180
```

实现了 Grafana 基于内存使用率的自动扩容

```
yanyu@master:~/桌面$ kubectl apply -f grafana-hpa.yaml
horizontalpodautoscaler.autoscaling/grafana-hpa created
yanyu@master:~/桌面$ kubectl get hpa -n monitoring
NAME          REFERENCE            TARGETS                        MINPODS   MAX
PODS   REPLICAS   AGE
grafana-hpa   Deployment/grafana   <unknown>/40%, <unknown>/70%   1         3
       0          9s
yanyu@master:~/桌面$ kubectl describe hpa grafana-hpa -n monitoring
Name:                                                  grafana-hpa
Namespace:                                             monitoring
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Wed, 07 Jan 2026 11:3
3:59 +0800
Reference:                                             Deployment/grafana
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):     3% (3m) / 40%
  resource memory on pods  (as a percentage of request):  161% (169267200) / 70
%
Min replicas:                                          1
Max replicas:                                          3
Behavior:
  Scale Up:
    Stabilization Window: 0 seconds
    Select Policy: Max
    Policies:
      - Type: Pods     Value: 4    Period: 15 seconds
      - Type: Percent  Value: 100  Period: 15 seconds
      - type: Pods  value: 1  Period: 60 seconds
Deployment pods:    3 current / 3 desired
Conditions:
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    ReadyForNewScale  recommended size matches current si
ze
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully ca
lculate a replica count from memory resource utilization (percentage of request
)
  ScalingLimited  True    TooManyReplicas   the desired replica count is more t
han the maximum replica count
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  2m27s  horizontal-pod-autoscaler  New size: 3; rea
son: memory resource utilization (percentage of request) above target
yanyu@master:~/桌面$ kubectl get pods -n monitoring -o wide | grep grafana
grafana-7bd56db9ff-jcnhm              1/1     Running   0              2m44s
 10.244.104.52     node2    <none>           <none>
grafana-7bd56db9ff-mkcsf              1/1     Running   4 (16m ago)    4d14h
 10.244.219.66     master   <none>           <none>
grafana-7bd56db9ff-mmklv              1/1     Running   0              2m44s
 10.244.166.142    node1    <none>           <none>
```

缩容成功

```
yanyu@master:~/桌面$ kubectl get pods -n monitoring | grep grafana
grafana-77658c89b9-7qpkc              1/1     Running   0              8m30s
grafana-77658c89b9-rwx62              1/1     Running   0              10m
yanyu@master:~/桌面$ kubectl get hpa grafana-hpa -n monitoring
NAME          REFERENCE            TARGETS           MINPODS   MAXPODS   REPLIC
AS   AGE
grafana-hpa   Deployment/grafana   7%/40%, 46%/95%   1         3         2
     5h33m
```

06

功能验证与结果展示

功能验证与结果展示 - 监控

Grafana仪表盘展示

节点CPU使用率

实时展示各K8s节点的CPU使用率,支持按节点筛选,可查看1小时/24小时/7天趋势图

节点内存使用率

展示节点内存总量、已用、可用及使用率,通过颜色区分正常/警告/危险状态

容器内存占用

按命名空间和Pod展示容器内存占用Top10,快速定位内存泄漏容器

Pod重启次数

统计各Pod的重启次数,帮助识别频繁重启的应用,便于问题排查

数据更新延迟

≤30s

监控数据更新延迟

- **采集间隔:** 15s
- **Prometheus处理:** 5s
- **Grafana刷新:** 10s

满足实时性要求

监控数据在半分钟内完成从采集到展示的完整链路

验证总结

- **可视化效果**
  仪表盘图表展示清晰
- **筛选功能**
  支持多维度数据筛选
- **查询性能**
  PromQL查询响应快速

功能验证与结果展示 - 日志

Loki日志查询界面

Grafana Explore界面

通过Grafana的Explore功能访问Loki日志查询界面,支持LogQL语法,提供强大的日志检索能力

功能特性:

- 实时日志流式展示
- 支持LogQL查询语法
- 日志数量统计分析

按Pod/命名空间筛选

按命名空间筛选

通过{namespace="monitoring"}语法筛选特定命名空间的日志,支持多命名空间联合查询

按Pod筛选

通过{pod=~"grafana-.*"}语法筛选特定Pod的日志,支持正则表达式匹配

按容器筛选

通过{container="main"}语法筛选特定容器的日志

关键词匹配查询

包含关键词

```
|= "error"
```

筛选包含"error"的日志

排除关键词

```
!= "info"
```

排除包含"info"的日志

正则匹配

```
|~ "(error|fatal)"
```

匹配error或fatal

日志延迟

≤45s

日志采集到查询延迟

延迟分解

Alloy采集:	10-15s
Loki处理:	5s
Grafana查询:	3s

功能验证与结果展示 - 告警

告警规则触发状态

NodeNotReady

FIRING

状态: 节点异常

时间: 2026-01-02 14:32

PodRestartFrequent

FIRING

状态: 频繁重启

时间: 2026-01-02 15:15

EasyMemoryAlert

FIRING

状态: 内存超限

时间: 2026-01-02 16:45

告警延迟

≤5min

告警触发延迟

延迟分解

指标采集:	~15s
规则评估:	~30s
for持续时间:	1-3min

实例级定位能力

节点级定位

NodeNotReady告警显示具体节点名称

Pod级定位

PodRestartFrequent告警显示具体Pod名称

资源级定位

EasyMemoryAlert告警显示具体节点和内存使用率

07

排错与复盘

排错与复盘 - 典型部分问题与解决方案

1 Loki连接失败

问题描述

Grafana 集成 Loki 数据源时，测试连接提示 502 Bad Gateway，无法读取日志数据

根因分析

Grafana 部署在 monitoring 命名空间，Loki 部署在 loki 命名空间，跨命名空间访问时未使用 Kubernetes 集群内部标准 DNS 地址，导致服务解析失败

解决方案

1. 修改 Loki 数据源访问地址为：‘http://loki-read.loki.svc.cluster.local:3100‘
2. 添加 HTTP 头 X-Scope-OrgID，值设为 ‘local’
3. 保存配置，测试连接成功

2 Alloy未实时采集新日志

问题描述

新创建的 Pod 日志未同步至 Loki，Grafana 中仅能查询旧 Pod 日志，Alloy 日志无明显报错

根因分析

Alloy 配置中日志采集规则未动态刷新，且网络限制导致部分 Pod 标签未被正确识别

解决方案

重启 Alloy 服务，确认未添加多余标签过滤条件，确保全集群 Pod 日志均能被采集

3 告警未触发

问题描述

模拟故障后,告警规则状态一直为Inactive,未进入Pending

根因分析

for时长设置过长

解决方案

调整for时长为30s-1m

4 Grafana卡顿

问题描述

Grafana界面加载缓慢,查询超时,响应延迟高

根因分析

单副本Grafana负载过高,无法处理并发查询请求

解决方案

将Grafana扩容至2副本,实现负载均衡和高可用

排错与复盘 - 资源与网络优化

Loki内存溢出

问题现象

Loki Pod频繁重启,状态为OOMKilled,日志显示内存不足

原因分析

Loki在处理大量日志时内存占用高,超出限制的1Gi

解决方案

删除内存限制配置,或调整至2Gi,重启Loki Pod

网络策略影响访问

问题现象

配置NetworkPolicy后,Prometheus无法抓取目标指标

原因分析

NetworkPolicy限制了Pod之间的网络流量,导致Prometheus访问被拒绝

解决方案

关闭NetworkPolicy或添加允许规则

镜像加速配置

优化前

拉取镜像耗时5-10分钟,Pod长时间处于ImagePullBackOff状态

原因分析

国内访问Docker Hub速度慢,影响部署效率

解决方案

配置国内镜像加速器,拉取时间缩短至30秒

优化效果总结

系统稳定性与性能显著提升

0	100%	10x
OOMKilled	网络连通	部署提速

08

小组分工与贡献说明

小组分工与贡献说明

成员编号	姓名	学号	主要任务	贡献率
成员 1	严羽	2362160013	1. 优化 Docker 配置 2. Prometheus+Grafana 部署与访问配置 3. 完成监控指标可视化 4. PPT 制作	25%
成员 2	张馨尹	2362160002	1. 设计 3 条告警规则 2. 进行故障模拟触发测试 3. 验证告警有效性 4. 实验报告撰写	25%
成员 3	张婉婷	2362160021	1. 部署 OpenEBS 存储 2. 部署 Loki 日志系统与 Alloy 采集端 3. 实现 Grafana 日志集成查询	25%
成员 4	肖思梦	2362160057	1. 执行 Grafana 扩缩容测试 2. 联调全流程闭环 3. 汇总实验文档与代码附件整理 4. 答辩	25%

团队协作模式

分工明确,优势互补,高效协同

项目完成度: 100%

所有目标均已达成

欢迎提问与交流

感谢聆听!

云计算技术 · 第一学期 · 23大数据1、2班

小组成员: 严羽 张馨尹 张婉婷 肖思梦

答辩日期: 2026-01-05